スケッチを用いた多次元データの高速類似検索に関する研究

著者	樋口　直哉
学位授与年度	令和元年度
学位授与番号	17104甲情工第345号
URL	http://hdl.handle.net/10228/00007777

スケッチを用いた多次元データの高速類似検索に関する研究

樋口 直哉

2020 年 3 月

概要

スケッチは，データをコンパクトなビット列で表現したもので，局所性鋭敏ハッシュ（Locality Sensitive Hashing, LSH）の一種である．本論文では，スケッチを用いた多次元データの類似検索について議論する．多次元データは距離空間に配置されていると仮定し，類似検索としては，k 近傍検索を用いる．ここで，k **近傍検索**とは，$k \geq 1$ に対し，与えられた質問データ q に対して，検索対象のデータベースから q に近い上位 k 個のデータを求めることである．

本論文では，**球面分割**（*BP*）に基づくスケッチを用いる．BP は，空間内のデータに対して，中心点が p，半径 r の球の内外に対応して，ビット “0” または “1” を割り当てる．この中心点と半径の対 (p, r) を**ピボット**という．ピボットを w 個用いて，長さ w ビットのスケッチを作成する．スケッチの長さを**幅**という．スケッチは，距離情報を部分的にしか保持できないので，スケッチを用いる k 近傍検索は 2 段階で行う．第 1 段階では，スケッチによる**フィルタリング**を行い，質問とのスケッチ間の距離が近い順に K 個の解候補を選択する．ただし，$K \geq k$ である．第 2 段階では，K 個の解候補に対してデータ間の実際の距離計算を行うことで k 近傍解を選択する．一般に，K を大きくすると正しい

解が得られる確率である**検索精度**は高くなるが，検索速度は遅くなる．また，スケッチの幅 w を大きくすると，フィルタリング性能は向上するが，フィルタリングのコストは大きくなる．

従来，第 1 段階検索での解候補選択のための優先順位付けとしては，スケッチ間の不一致ビット数を表す**ハミング距離**が用いられており，高い検索精度を保証するためには，スケッチ幅は 32 ビット以上が必要であると考えられていた．そのため，第 1 段階検索は，あらかじめ用意したすべてのデータのスケッチを格納したデータベースに対する全探索によって実現されてきた．第 1 段階で全探索を用いたとしても，スケッチが元のデータに比べるとコンパクトになっているので，元のデータを直接検索するよりも高速に実行できる．

本研究では，まず，ビット幅を固定した場合のフィルタリング性能が高くなるようにスケッチを設計する最適化手法について議論する．スケッチのフィルタリング性能を直接測定するとコストが大きいので，その代わりに，最適化の目的関数としてスケッチの衝突確率を用いて，その最小化を行う．**衝突**とは，異なるデータに対して同じスケッチが割り当てられることをいう．データベース内のデータ分布を考慮する発見的手法として，2 値量子化法を提案し，有効であることを示す．また，焼きなまし法の効率的実現法である AIR(Annealing by Increasing Resampling) による最適化も有効であることを示す．衝突確率を下げていくと，フィルタリング性能が向上する傾向は確認できるが，衝突確率を下げ過ぎると，必ずしもフィルタリング性能が最大となるとは限らないことも確認できる．

つぎに，スケッチを**次元縮小射影**とみなすことにより，質問とスケッチ間の**距離下限**を求めることができることを示し，距離下限の集計で定義される新しい優先順位付け $score_\infty, score_1, score_2$ を提案し，第 1 段階の解候補の選択基準としてハミング距離よりも高性能である，つまり，高精度のフィルタリングが可能となることを示す．

さらに，従来よりも幅が狭い 16 ビット前後のスケッチを用いることにより，バケット法を用いたデータ管理による高速化を可能とし，質問との優先順位順にスケッチを**列挙**することにより第 1 段階の検索コストをほとんど無視できる手法を提案する．16 ビット前後の幅のスケッチを用いる検索は，精度を維持するためには，32 ビットスケッチを用いる場合より大きな候補数 K を必要とする．しかし，バケット法ではデータオブジェクトをスケッチの値によってソートして，第 2 段階の検索におけるメモリ局所性を向上することにより，候補数 K の増加による速度低下を低減できる．提案手法を用いると，約 700 万件の画像特徴データ（64 次元）や音特徴データ（96 次元）を用いた最近傍検索実験により，16 ビットスケッチと $score_1$ による優先順の列挙を用いる提案手法では，32 ビットスケッチとハミング距離を用いる従来手法より，90% 以上の検索精度が要求される場合でも 20 倍以上の高速化が実現できることがわかる．なお，この検索速度は，スケッチを用いない全探索と比べると 50〜100 倍程度高速である．

また，データベースのデータ件数やデータの次元数と最適なスケッチ幅の関係について調査し，データ件数が大きくなるとスケッチ幅を広くする方がよく，データの次元数が高くなるとスケッチ幅は狭くした方がよいという傾向があることを示す．

謝辞

本研究を遂行するにあたり，最終講義を終えながらも変わらずご指導を続けて頂いた九州工業大学の篠原武名誉教授，主査としてご指導頂きました九州工業大学情報工学研究院の平田耕一教授，学外からご指導頂いた学習院大学計算機センターの久保山哲二先生と株式会社 THIRD チーフ・サイエンティフィック・オフィサーの今村安伸さんに心よりお礼申し上げます．また，副査 として審査を通してご指導頂きました九州工業大学情報工学研究院の坂本比呂志教授と九州工業大学情報工学研究院の久代紀之教授にも大変お世話になり，感謝致しております．最後に，精神的にも経済的にも私の学生生活を終始支えて頂いた両親に感謝致します．

目次

第 1 章

はじめに

1.1　多次元データの検索と次元縮小

計算機の演算能力や記憶容量の向上により，大量のマルチメディアデータを用いたシステムが多く作られている．そのため，膨大なデータの中から必要なデータだけを高速に探し出す情報検索技術が重要である．マルチメディアデータは多くの場合でかなり高次元であり，また劣化や加工も多く，完全一致検索によって目的を達成することは難しい．そのため，それらのデータの高速な検索を実現するために，類似検索を用いることが一般的である．

本論文では，距離に基づいた類似検索を取り扱う [28]．データ間の非類似性は距離によって定義されていると仮定し，距離が近いものを似ていると考える．また，類似検索の手法としては，もっとも代表的な近傍検索について議論する．類似検索の一般的な検索手法として，R-tree [4, 24] や M-tree [1] などの階層的空間索引構造を用いるものがある．

それらの階層的空間索引を用いた場合でも，空間の次元が大きくなると次元の呪いと呼ばれる現象により，検索効率が悪化してしまうことが知られている．そのため，高次元のデータに索引構造を用いる場合，次元縮小を行うことで次元の呪いを緩和する．しかし，空間索引を用いた検索では，データベース内に類似データが存在する質問に対しては非常に高速に検索できる反面，類似データが存在しない質問に対しては検索速度が遅くなる．

本論文では，多次元空間における効率的**類似検索**を実現するために，**スケッチ** [23, 19, 2, 17, 26, 18, 20, 16, 21, 22, 6, 13, 14] を用いる．スケッチは，多次元データを表したコンパクトなビット列であり，データ間の類似性をある程度保持できる局所性鋭敏ハッシュ（Locality Sensitive Hashing, LSH）の一種である．スケッチは，**球面分割**（Ball Partitioning，BP）や一般化超平面分割（Generalized Hyperplane Partitioning，GHP）などの基礎分割関数を用いて空間を任意の回数分割することで作成される．BP は，データに対して，球内であればビット 0，そうでなければビット 1 を割り当ててスケッチを作成する方法である．BP は，Vantage-point tree [27] でも用いられている．

高速な類似検索の応用例として，動画同定や音楽同定などが考えられる．例えば，Youtube などの動画投稿サイトにおける違法アップロードされた動画の検出に用いることができる．この場合，保護対象の動画のコマ画像を特徴抽出したデータをデータベース，違法アップロードされた動画か調べたい動画のコマ画像を特徴抽出したデータを質問として検索することで，質問と距離が近いデータがある場合にはその質問をもつ動画は違法アップロードされた動画の可能性が高いことがわかる．

1.2　スケッチを用いた類似検索

スケッチを用いた類似検索は 2 段階からなる．第 1 段階では，スケッチ間の距離に応じて候補を選択する．第 2 段階では，元の空間内の距離を使用して，質問と候補を比較することにより，解を選択する．スケッチを用いた検索では，スケッチ間の距離がデータ間の距離を完全に反映することができないため，階層的空間索引 R-tree [4] や M-tree [1] を用いた検索と異なり，正確な解を求めることができない．階層的空間索引法と遜色ない速度を保ちつつ，ある程度の精度を保証するためには，スケッチのビット数は 32 ビットや 64 ビット必要であると考えられてきた．スケッチの幅が広いほど検索精度が向上するが，同時に検索コストも増大する．反対に検索コストを削減するためにスケッチの幅を狭くすると，高い検索精度を達成するために必要な解候補数が多くなる．そのため，幅が広すぎず，かつ高い検索精度を達成できるスケッチの作成が重要である．

1.3　研究成果の概略

1.3.1　スケッチの最適化

スケッチを用いる検索の精度向上ためには，ピボットの選択における最適化が重要である．従来の最適化では，ピボット候補としてデータベースからランダムにデータを取り出し，そのピボット候補を評価関数によって評価する．同様の処理を任意の回数行い，最も評価が高いピボット候補をピボットとして決定することで行っていた．ピボッ

トの評価には最小衝突法を用いている．本来異なるデータがスケッチ空間上において同じバイナリ文字列になることを**衝突**と呼び，この衝突が少ないほど良いピボットと見なす評価手法である．基礎分割関数には，データベース内のデータ分布を考慮する発見的手法として，2 値量子化法を提案し，有効であることを示す [11]．また，焼きなまし法（Simulated Annealing, SA）の効率的実現法である増加再標本焼きなまし法（Annealing by Increasing Resampling, AIR） [12, 5] による最適化も有効であることを示す．AIR は，評価に標本からランダムに選んだ再標本を用いて，従来の SA における温度制御を評価用の再標本サイズ制御に置き換えている．SA の高温時には AIR の小さい再標本サイズ，低温時には大きい再標本サイズを対応させているので，AIR は SA に比べて高速に最適化することができる．

衝突を減らすためには，データを半々に分ける BP がよいことは明らかであるが，フィルタリング性能については必ずしも向上せず，アンバランスなスケッチの方がよい場合がある [17]．本研究で行った実験結果でも，AIR でピボット選択を行うと衝突確率はある程度小さくできるが，衝突率を下げ過ぎると，却ってフィルタリング性能が悪くなる場合があることが確認できる．

1.3.2　スケッチによる解候補選択基準の高精度化

本論文では，第 1 段階における解候補選択基準として，ハミング距離の代わりに，距離下限値に基づくスコアを用いて，検索精度・速度を向上することができる手法を提案する [7]．これは，球面分割によるスケッチは次元縮小射影 Simple-Map [25, 15] の量子化

像とみなすことができるという事実に基づいた手法である．Simple-Map は．ピボットの点 p を用いて，データ x を p との距離 $D(p,x)$ に射影する．この射影像を r を閾値として 2 値に量子化するとスケッチを構成する球面分割になる．Simple-Map による射影像間の距離は，2 値量子化の後では，量子化誤差が大きくなり，役に立たない．しかし，実際には質問については，量子化せずにそのまま Simple-Map で射影できるので，質問の像については量子化せずにそのまま用いれば，量子化誤差をある程度復元できる．このことを利用して，質問とスケッチ間の距離下限を求めることができる．距離下限値は質問と分割境界との最小距離によって求められる．距離下限値は質問とピボットで計算されるため，データベースのデータごとに再計算する必要がないので計算コストは少ない．また，この距離下限値に基づくスコアの計算は，質問とデータのスケッチが一致している場合は 0，そうでない場合には距離下限値を用いて，その最大値を求める $score_\infty$ や総和を求める $score_1$ などがある．このスコアの計算は，ビットパターンに対する部分スコアを表（配列を用いる）関数とすることで高速に計算することができるので，ハミング距離の距離計算と比べてもほとんど変わらない検索速度を達成できる．

同様の距離下限値に基づく方法が過去に提案されている [2]．この論文では，本論文とは異なりスケッチを次元縮小として用いていない．また，スケッチの幅が広く，次元数が減っていない（むしろ次元数が大きくなる場合が多い）ため，距離下限値に基づくスコアの計算コストが実距離計算コストよりも大きくなるという問題があった．そのため，このスコアをハミング距離と置き換えて用いるのではなく，ハミング距離によって大雑把に解候補を絞り込んだあと，スコアによって解候補を決定するという 2 段階でフィルタリング

を行っている．この点で，従来のハミング距離と同程度の時間でスコア計算できる本論文の手法の方が優れている．

1.3.3　ビット幅の狭いスケッチの利用

本論文では，より幅の狭い 16 ビットのスケッチを用いる手法を提案する [9, 10]．データベースの大きさは，数百万であると仮定する．32 ビットのパターンの個数は 2^{32}=約 43 億個あり，データベースサイズがそれを超えるほど巨大でない限り，空のバケットが多過ぎて非効率になるし，そもそも単純に配列を用いたバケット表は超巨大になるので現実的でない．32 ビットより幅が広いスケッチを用いる場合は，第 1 段階検索は，データのすべてのスケッチをそのまま用意しておき，質問のスケッチとの距離を計算して，全探索により解候補を選択する．

それに対し，16 ビットのパターンの個数は 2^{16} 個しかないので，スケッチをキーとするバケット法でデータを管理することができる．そうすると，第 1 段階検索は，2^{16}=約 6 万 6 千個のスケッチとの照合を行えばよく，データベースサイズに依らない一定コストで高速に実行することが可能となる．また，16 ビットスケッチの第 1 段階検索の候補選択に用いられる質問のスケッチと近いスケッチは，約 6 万 6 千個のうちのごくわずかでしかない．したがって，近い順にスケッチを列挙するアルゴリズムを利用すれば，スケッチ間の照合を行う必要がなくなり，事実上コストを無視できるほど第 1 段階検索を高速化することが可能である．

バケット法とは，データにキーを割り当てておくことで任意のデータに瞬時にアクセス

することが可能な手法である．図 1.1 に，8 個のデータからなるデータベースとそのキー，またそれぞれのキーに対応したバケツ（配列）を示している．例として，キー 7 のデータにアクセスすることを考える．全探索する場合は全てのキーを照合し，キー 7 のデータを取り出す．バケット法を用いる場合は，$bucket[7]$ とすることでデータベースを照合することなく瞬時に目的のデータを得ることができる．スケッチを用いる場合は，データベースをあらかじめスケッチ順にソートしておく．スケッチは整数とみなせるので，スケッチを配列の引数（キー）に与える．このとき，配列の値はそのスケッチであるデータ群の先頭アドレスとすることで，同じスケッチのデータを連続で参照できる．

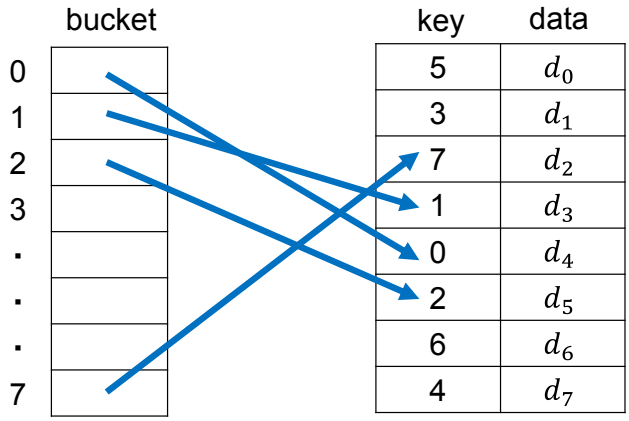


図 1.1　バケット法の例

1.3.4　スケッチの列挙を用いた照合不要の高速検索

ここで，スケッチの列挙による第 1 段階検索の高速化 [8] について，概略を説明しておこう．検索前にすべての 16 ビットパターンを ON ビットの個数の昇順にソートしたものを準備しておく．質問とこのビットパターンとのビットごとの排他論理和（XOR）を求

めると質問とのハミング距離の順にスケッチを列挙することができる．この列の先頭部分のみを用いて第 2 段階検索を実行する．この手法により，第 1 段階検索では，スケッチ間のハミング距離計算による照合が不要となり，検索コストをほとんど必要としなくなる．また，データベースのデータをスケッチ順にソートしておくことで，第 2 段階の検索におけるメモリ局所性を向上できる．

例を用いて，第 1 段目の高速化のためのスケッチの列挙法を説明しよう．スケッチの幅を 3 ビットとする．スケッチ間の距離はハミング距離を用いるとする．スケッチ間のハミング距離は，0, 1, 2, 3 の 3 種類である．質問のスケッチ $s = 011$ とすると，それぞれの距離となるスケッチは，以下の通りである．

距離 0 のスケッチ

s と一致するスケッチは 1 個，s 自身，つまり，011.

距離 1 のスケッチ

s と 1 ビットだけ異なるスケッチは 3 個あり，1 箇所だけ 1 の 3 ビット列（001, 010, 100）と s との XOR で求められる．それぞれ，$s \oplus 001 = 010$, $s \oplus 010 = 001$, $s \oplus 100 = 111$ である．

距離 2 のスケッチ

s と 2 ビットだけ異なるスケッチ 3 個は，2 箇所だけ 1 の 3 ビット列（011, 101, 110）と s との XOR で求められる．それぞれ，$s \oplus 011 = 000$, $s \oplus 101 = 110$, $s \oplus 110 =$101 である．

距離 3 のスケッチ

s と 3 ビットすべて異なるスケッチは 1 個．3 箇所すべて 1 の 3 ビット列 111 と s の XOR で求められる．$s \oplus 111$ =100

このように，3 ビットスケッチの場合には， ON ビット数の昇順のビットパターンの列 000, 001, 010, 100, 011, 101, 110, 111 と質問のスケッチとの XOR を取ることで質問とのハミング距離の昇順にスケッチを列挙することができる．XOR に用いるビットパターン列は質問に依らないので，ON ビットの個数の昇順に並べたものを事前準備しておくことができる．データベースのデータはスケッチをキーとしたバケット法で管理することができるので，第 1 段階の検索においては，データと質問のスケッチ間の距離を計算する必要がない．これにより，事実上，第 1 段階の検索コストは無視することができる．

検索優先順位付け $score_\infty$ や $score_1$ に対しても，スケッチを列挙するアルゴリズムを提案する．それぞれの質問に対して，スケッチの各ビットに対応する分割境界との最小距離の表を用意し，それらの順位を求めておけば，質問とのスコアの小さい順にスケッチを列挙することができる．それを利用して，スコアの小さい順にデータが K 個になるまで第 2 段階検索を実行することができる．各質問のための準備のためのコストは無視できるほど小さいので，実際には，第 1 段階目の検索コストは，ハミング距離のときと同様に，事実上無視できる．

実際の画像データベースや音楽データベースを用いた最近傍検索の実験により，90% 程度の高精度が要求される場合でも，16 ビットスケッチを用いた提案手法は，従来の 32

ビットスケッチを用いた手法より 10 倍程度高速であり，素朴な全探索法より 100 倍程度高速であることを確認することができる．

第 2 章

準備

以下の議論で用いる概念について簡単に説明しておく．多次元データの類似検索では次元の呪いによって検索性能が悪化することが知られている．そのため低次元に射影する次元縮小が重要である．本論文では，次元縮小射影としてスケッチを用いる．スケッチは基礎分割関数によって空間を分割し，それぞれの部分空間にビット “0”，“1” を割り当てる．基礎分割関数を任意の回数適用することで任意の幅のスケッチを作成する．本論文では，基礎分割関数として球面分割を用いる．球面分割に基づくスケッチは次元縮小射影 Simple-Map の像を量子化したものとみなすことができる．

2.1 距離に基づく類似検索

本論文では，**類似検索**とは，データ間の非類似度 (距離) を用いて，質問点と類似するデータを取り出すことである [28]．検索対象のデータベースの特徴空間全体を $U = \mathbb{R}^m$

とする．ここで，$\mathbb{R}$ は実数全体，m は特徴データの次元数である．任意の 2 点のデータ間の非類似度の指標を示す距離関数を $D : U \times U \to \mathbb{R}^+$ とし，(D, U) を距離空間とする．本論文で扱う類似検索では，距離関数 D は距離の公理である次の条件を満たすものとする．

1. $D(x, y) \geq 0$ (非負性)
2. $D(x, y) = D(y, x)$ (対称性)
3. $D(x, y) \leq D(x, z) + D(z, y)$ (三角不等式)
4. $D(x, y) = 0 \Leftrightarrow x = y$ (同一性)

ここで，$x, y, z \in \mathcal{U}$ である．上の条件の中で最も重要なものは，三角不等式である．

本論文では類似検索手法としては最近傍検索，つまり，$k = 1$ のときの k 近傍検索について議論する．ここで，最近傍解が複数あった場合でもそのうちの一つだけを得られればよいとする．与えられたデータベースのデータは，0〜$n-1$ の自然数で索引付けされているとする．n 個のデータからなるデータベースを

$$db = \{x_0, \ldots, x_{n-1}\} \subseteq U$$

とする．2 個のデータ x_i と x_j 間の非類似度は，距離 $D(x_i, x_j)$ で定義される．質問 $q \in \mathcal{U}$ に関する**最近傍検索**とは，すべての $y \in db$ に関して $D(q, x) \leq D(q, y)$ となるような $x \in db$ を見つけることである．

次に，本論文で用いる距離計測について説明する．特徴空間内の任意のデータを x とす

る．x の特徴は m 個の実数の組 $(x_{(0)}, x_{(1)}, \ldots, x_{(m-1)})$ で表される．以下の三つの距離計測関数は距離の公理を満たす．

$$L_1\text{距離} : D(x,y) = \sum_{i=0}^{m-1} |x_{(i)} - y_{(i)}| \tag{2.1}$$

$$L_2\text{距離} : D(x,y) = \sqrt{\sum_{i=0}^{m-1} (x_{(i)} - y_{(i)})^2} \tag{2.2}$$

$$L_\infty\text{距離} : D(x,y) = \max_{i=0}^{m-1} |x_{(i)} - y_{(i)}| \tag{2.3}$$

2.2 次元縮小

次元縮小の必要性について概略を説明する．次元縮小は，射影後の距離は縮むことはあっても伸びないものとする．多次元データの類似検索では次元の呪いによって検索性能が悪化することが知られている．そのため低次元に射影する次元縮小が重要である．従来法として主成分分析法や FastMap [3]，Simple-Map [25] などがある．本論文では，次元縮小射影として球面分割によるスケッチを用いる．球面分割によるスケッチは，Simple-Map の像を 2 値量子化したものとみなすことができるので，まず，Simple-Map を簡単に紹介する．

2.3 次元縮小射影 Simple-Map

Simple-Map（S-Map）は射影関数として，中心点とデータの実距離を用いる．また，S-Map は距離の公理を満たすすべての距離空間に対して適用可能である．射影空間上では任意の 2 点間の距離が，元空間の距離よりも伸びることはないが縮まる可能性がある．このことを**距離の縮み**と呼ぶ.

射影に用いる中心点を $p \in U$ とすると，任意のデータ $x \in U$ の射影空間上での座標値は

$$\varphi_p(x) = D(p, x)$$

と定義される．三角不等式より，任意の 2 点 x，y 間の射影空間における距離を $D'(x, y)$ とすると，射影距離 $D'(x, y)$ は中心点を媒介することで距離の縮みが生じる可能性がある．

$$D'(x, y) = |\varphi_p(x) - \varphi_p(y)| \leq D(x, y)$$

中心点を複数用いた場合に拡張する．S-Map による射影では，中心点の数が射影空間での次元数となる．中心点の集合を $P = \{p_0, p_1, \ldots, p_{m'-1}\}$ とすると，射影関数は

$$\varphi_P(x) = (\varphi_{p_0}(x), \varphi_{p_1}(x), \ldots, \varphi_{p_{m'-1}}(x))$$

となる．ここで m' は射影空間での次元数である．また，複数の中心点を用いて射影され

た 2 点間の距離は，

$$D'(x,y) = \max_{p \in P} |\varphi_p(x) - \varphi_p(y)| \leq D(x,y)$$

となる．このように，射影空間上での距離は L_∞ 距離で計測する．中心点の数が多いほど，射影空間上で距離の情報を多く保存できる．つまり中心点を多くとることで距離の縮みを小さくできるが，射影空間が高次元になるため，射影距離のコストが増加し次元の呪いの影響を強く受ける．

2.4 球面分割に基づくスケッチ

本論文では**球面分割**に基づくスケッチを用いる．中心点と半径のペア (p, r) を**ピボット**と呼ぶ．球面分割 BP は，以下のように定義される．

$$BP_{(p,r)}(x) = \begin{cases} 0, & \text{if } D(p,x) \leq r, \\ 1, & \text{otherwise.} \end{cases}$$

BP に基づく**幅** w のスケッチ関数 s_P は w 個のピボット集合

$$P = \{(p_0, r_0), \ldots, (p_{w-1}, r_{w-1})\}$$

を用いて，以下のように BP によるビットを連接したものとして定義される．

$$s_P(x) = BP_{(p_{w-1},r_{w-1})}(x) \cdots BP_{(p_0,r_0)}(x)$$

二つの長さ w のビット列 s と t の**ハミング距離**は，不一致ビットの個数である．このハミング距離は，s と t のビット毎の排他論理和を求めて，その中のビット“1”の個数

で求めることができるので，ビット演算子を用いて高速に求めることができる．また，w ビットのビット列を w 次元の単位超立方体の 2^w 個の頂点に対応させることができるが，ハミング距離は，そのときの L_1 距離に一致している．

スケッチは，元のデータをよりコンパクトに表現しているが，ハミング距離を用いる限り次元縮小とみなすことはできない．本論文では，従来のハミング距離よりも高精度な検索優先順位付けとして，質問とスケッチ間の距離下限値に基づくスコアによる優先順位付けを提案する．これについては，第 4 章で詳しく議論するが，球面分割に基づくスケッチが Simple-Map の像を量子化したものとみなすことができるという事実に基づいている．

2.5 スケッチを用いた最近傍検索

スケッチは，通常の用い方では，ハミング距離を用いる限り，厳密な意味では次元縮小ではない．しかし，本論文では，後に第 4 章で述べるように，スケッチを次元縮小の一種として扱う．スケッチはデータ間の距離を部分的にしか保持できないので，スケッチを用いた最近傍検索は，以下のよう 2 段階で行う．ここで，s はデータをスケッチに射影する関数とし，$K \geq 1$ はユーザが指定するパラメータとする．

1. 準備段階:

   すべてのスケッチ $s(x_0), \ldots, s(x_{n-1})$ を計算する．

2. 第 1 段階 （スケッチ間の距離によるフィルタリング）:

   質問 q のスケッチ $s(q)$ に近いスケッチ $s(x_{i_0}), \ldots, s(x_{i_{K-1}})$ をもつ K 個の解候補

$x_{i_0}, \ldots, x_{i_{K-1}}$ を選ぶ.

3. 第 2 段階 （実距離計算による最近傍検索）:

   解候補 $x_{i_0}, \ldots, x_{i_{K-1}}$ から最近傍データを選ぶ.

スケッチは，オリジナルの特徴データに比べて比較的小さな構造である．たとえば，本論文の実験では，64 バイトの画像特徴データに対して 32 ビットや 16 ビットのスケッチを使用する．検索の第 1 段階において，スケッチ間の距離としては，従来は，特徴間の実距離よりもビット演算によって容易に計算できるハミング距離が用いられている．しかし，スケッチ間の距離は元の距離関係を完全に保存できないので，近いものを解候補として選ぶためのフィルタとして用いる．フィルタリングの選択基準に用いられる距離を**検索優先順位付け**と呼ぶ．検索の**精度**は，正しく最近傍が求められる確率である．スケッチを用いる検索では，第 1 段階で求める K 個の候補に最近傍が含まれている確率である．第 1 段階における解候補数 K が大きいほど精度は上がるが，検索は遅くなる．したがって，スケッチにおける最も重要な課題の一つは，より小さい K で高精度を達成する，あるいは，許容可能な誤差で検索速度を上げることである.

2.6 関連研究

ここで，本研究と特に関連が深い研究として Dong らの論文 [2] と比較を行う．この論文は，距離下限値に基づくスコアを提案しているという点で本研究と類似している．表 2.1 に本研究との比較を示す.

表 2.1　関連研究との比較

	本研究	Dong ら [2]
基礎分割関数	球面分割	L_1 スケッチ
特徴データ数	700 万	450 万
特徴データ次元数	64	128
スケッチのビット幅	16〜32	100〜1000
スケッチの用い方	次元縮小	データサイズ削減
スケッチの最適化	あり	なし
距離下限値の使い方	検索優先順位付け	距離の見積もり
適用範囲	任意の距離空間	座標系をもつ距離空間
フィルタリング手法	スケッチの列挙（照合なし）	データのスケッチとの照合（全探索）

本研究との主な違いはスケッチの用い方にある．Dong らの研究では，次元縮小ではなく，元の特徴データのデータサイズが削減できればよいというスタンスである．そのため，スケッチの次元数（ビット幅）は元の特徴次元数と同等以上になっている．それに対し，本研究では次元縮小としてスケッチを用いているので，元の特徴次元数よりもスケッチの次元数は小さい．この違いが距離下限値の使い方に差を与えている．Dong らの研究では，スケッチの次元数が元の特徴次元数と同等以上であるために，距離下限値の計算コストが特徴距離計算コストよりも大きくなる傾向にある．そのため，距離下限値を従来のハミングの代替として用いるのではなく，1 段階目でハミング距離によって解候補をおおよそ絞り，2 段階目で距離下限値に基づくスコアによって 1 段階目で絞った候補から最終的な解候補を選ぶという 2 段階で処理している．それに対し，本研究ではスケッチの次元数が元の特徴次元数よりも小さいため，距離下限値に基づくスコアの計算コストも小さくできる．このため，従来のハミングの代替として距離下限値に基づくスコアを用いること

で解候補を選択することが可能である．

2.7 実験で使用するデータ

本論文では，検索対象となるデータとして，以下のような画像データや音楽データを用いた実験結果を示す．

- 画像データ: 2,900 本の動画のコマ画像から 2 次元周波数スペクトラムとして特徴抽出した約 700 万件の 64 次元データ
- 音楽データ: 1,400 本の音楽 CD から切り出した短い音データをメル周波数軸変換によって特徴抽出した約 700 万件の 96 次元データ

いずれの特徴データも周波数強度の対数を 8 ビット符号なし整数で表しており，総和が一定になるように正規化している．文献 [7] での実験においては，SISAP の Colors データセットも用いていた．しかし，Colors は，データが約 10 万件しかなく，本論文では数百万件のデータを仮定しているので用いていない．

また，画像データについては表 2.2 ようにデータ数，特徴次元数を減らしたものを用意して，提案手法とデータ件数や特徴次元数の関係を調査するために使用する．データ数削減については，動画の本数を変えずに動画の連続したデータを間引くことで 1/4，1/16 のデータベースを作成した．特徴次元数については，ローパスフィルタに相当する低周波数成分を取り出すことで $6 \times 6 = 36$，$4 \times 4 = 16$ 次元のデータベースを作成した．

ランダム生成されたデータは高次元空間において近くにデータが存在することがほとん

表 2.2　データ数や特徴次元数の異なる画像データベース

データベース	Database 1	Database 2	Database 3	Database 4	Database 5
データ数 ($\times 10^6$)	6.9	6.9	6.9	$6.9 \times \frac{1}{4}$	$6.9 \times \frac{1}{16}$
特徴次元数	64	36	16	64	64

どないので，最近傍検索の実験で用いる質問には不適切である．画像データベース中のデータ間の距離は，平均 1,650 である．ランダムデータと最近傍の距離は，平均 3,300 である．これに対して，データベース中の動画を低画質化したものから作成した質問の最近傍との距離は，300 前後であり，データベース中に似たものがない動画から作成した質問の最近傍との距離は，平均 600 程度である．したがって，本論文での実験では，データベースからランダムに選んだ 2 個のデータ x と y を用いて，x に対して，ノイズとして y を 5%，10%，...，50% の割合で混ぜた質問を作ることにした．たとえば，ノイズレベル 5% の質問 q はデータベースからランダムに選んだデータ x と y をそれぞれ 95% と 5% の重みで加重和したものである．つまり，

$$q = 0.95x + 0.05y$$

である．それぞれのノイズレベルについて，1,000 質問作る．質問のレイズレベルごとの最近傍距離の平均を図 2.1 に示す．質問をノイズレベルによって 5 個のグループ $5-10\%$，$15-20\%$，$25-30\%$，$35-40\%$，$45-50\%$，に分け，それぞれ，very-near，near，middle，far，very-far とする．

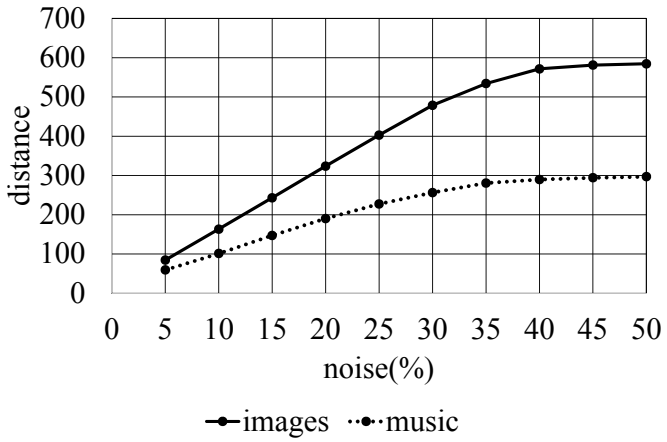


図 2.1　ノイズを混ぜた質問の平均最近傍距離

第 3 章

スケッチの最適化

スケッチを用いた検索では，フィルタリングの性能を改善することによって，精度や速度を向上することが重要である．ここでは，基礎分割関数の選択によって，スケッチを最適化することについて議論する．スケッチのフィルタリング性能は，測定コストが大きいので，最適化の目的関数としては不向きである．そこで，異なるデータに同じスケッチが割り当てられる衝突確率を目的関数として，スケッチの最適化を行う．最適化の手法としては，球面分割のピボットの中心点は，データの存在範囲の外側にとる方がよいという経験則とデータの分布を利用する 2 値量子化法 QBP を用いる．また，焼きなまし法（Simulated Annealing, SA）の効率的な実現法である増加再標本焼きなまし法（Annealing by Increasing Resampling, AIR）による最適化も試みる．本章は，論文 [11, 12] に基づいている．

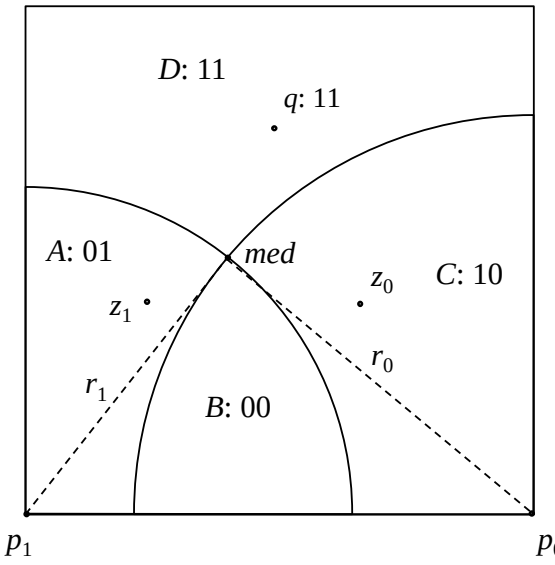
3.1 最小衝突法と量子化球面分割（QBP）

スケッチを用いた検索の精度を高めるために，衝突確率が小さくなるようにピボット集合 P を選ぶ．異なるデータ x と y に対して，それらのスケッチが一致するとき，すなわち，

$$s_P(x) = s_P(y)$$

となるとき，**衝突**が発生するという．本論文では，まず，量子化球面分割（Quantization Ball Partitioning, QBP）を用いてスケッチの衝突が少なくなるように最適化を行う．数百万件のデータベースに対して，最適化された 16 ビット程度の比較的幅の狭いスケッチを用いる場合は，各スケッチに射影されるデータ数はある程度均等になることが期待できる．

スケッチを用いた検索では用いる基礎分割関数によって，検索性能が大きく変化する．中心点をデータベースから選んだ BP は中央値付近のデータの多くが球の内側となってしまうため，衝突が多くなってしまうことが考えられる．衝突を少なくするには，ピボットの中心点を空間の隅に取ることが有効であるが，空間の隅は次元数に対して指数的に多く存在するので，素朴な方法で効率的に選択するのは困難である．本研究では，こうした問題を解決できる QBP を用いる．QBP は，データベースから任意に点を選び，データ中央値を閾値として空間の最小値もしくは最大値の 2 値に量子化して，ピボットの中心点とする．これにより，データ分布を考慮した効果的なピボット選択が可能となる．



部分空間	ハミング距離
A	1
B	2
C	1
D	0

質問 q からの優先順位（ハミング距離）

$D < A = C < B$

図 3.1　QBP による 2 ビットスケッチとハミング距離

Algorithm 1 に QBP を用いて，衝突確率が小さなピボット集合を求める発見的手法 SelectPivotQBP を示す．SelectPivotQBP は，1 ビットスケッチから始めて，幅を順次増やしながら追加するピボットをパラメータ *Trials* で指定された回数乱択により衝突確率が小さくなるように選んでいる．

図 3.1 に，2 次元ユークリッド平面上における QBP による 2 ビットスケッチとハミング距離による優先順位の例を示す．データの中央値を med とする．データからランダムに選んだ 2 点を z_0，z_1 とすると，med を閾値として 2 値量子化すると，それぞれ，med より左下，右下にあるので，p_0，p_1 になる．量子化した点と中央値の距離を r_0，r_1 として，2 個のピボット集合 $P = \{(p_0, r_0), (p_1, r_1)\}$ を用いると，2 ビットスケッチが得られる．中央値との距離を半径とするのは，球面分割によってデータがほぼ半々に分けられるからである．2 個の球面分割によって，平面は，4 個の部分空間 A, B, C, D に分けられ

```
// m: 特徴データの次元数, n: データベースのデータ数
// x[0], x[1], ..., x[n − 1]: データベースの特徴データ
// z[j]  (j = 0, ..., m − 1): データ z の第 j 次元の特徴値
// med[j]  (j = 0, ..., m − 1): データベースの特徴データの第 j 次元の中央値
// Trials: 各ピボットの探索の試行回数
// random(n): 0〜n − 1 の一様乱数
// MIN, MIN: 特徴値の最大値と最小値
1  procedure SelectPivotQBP(p[w][m], r[w], w)
       // p[w][m], r[w]: ピボットの中心と半径のための配列
       // w: スケッチの幅（ビット数）
       // E: ピボット群の標本 S に対する評価値（衝突確率）を求める関数
2      for i = 0 to w − 1 do
3          best ← ∞;
4          for t = 1 to Trials do
5              c ← random(n); z ← x[c];
               // 2 値量子化（binary quantization）
6              for j = 0 to m − 1 do
7                  if z[j] ≤ med[j] then
8                      p[i][j] ← MIN;
9                  else
10                     p[i][j] ← MAX;
11             r[i] ← D(p[i], med);
12             current ← E({(p[0], r[0]), ..., (p[i], r[i])});
13             if current < best then
14                 best ← current; temp ← p[i]; rtemp ← r[i];
15         p[i] ← temp; r[i] ← rtemp;
```

Algorithm 1: 2 値量子化を用いたピボット選択アルゴリズム SelectPivotQBP

る．A または B, C, D の任意の点 x に対して，$s_P(x)$ は，それぞれ 01 または $00, 10, 11$ である．図のように，q を両方の球の外側の質問とすると，q からの部分空間の任意の点に対するハミング距離による優先順位は，図の右の表のようになる．ここで，ハミング距離では部分空間 A と C が区別できないがことに注意しよう．

3.2 焼きなまし法によるスケッチの最適化

本節では，増加再標本焼きなまし法（Annealing by Increasing Resampling, AIR）[11, 12] を用いたスケッチの最適化について議論する．ここでは，X を解空間とし，X の要素を**状態**という．評価用のデータの母集団を F とし，その有限部分集合 $S \subseteq F$ を**標本**という．**目的関数**を $E : X \times 2^F \to \mathbb{R}$ とし，$E(x, S)$ は標本中の個々の $y \in S$ に対する評価の平均で与えられていると仮定する．ここでは，**最適化**は，$E(x, S)$ を最小とする $x \in X$ を求める最小化問題であるとする．また，任意の状態 x の**近傍** $Nb(x) \subseteq X$ が定められていると仮定する．

従来の SA では，温度を用いて高温時にランダムウォーク，徐々に温度が下がるにつれて局所探索を行うようにする．Algorithm 2 に SA の概略を示す．ただし，T_t は，時刻 t における温度，$Pr(T, \Delta E)$ は温度 T のときに評価差が ΔE である状態に遷移する**受理確率**である．

これに対して，AIR は目的関数の評価に再標本を用い，温度の代わりに再標本サイズを変化させることで最適化を行う．高温時にはサイズが小さい再標本を用いることで誤評価を発生させ，従来の SA におけるランダムウォークを実現しており，低温時には標本全

```
1 procedure SA
    // random(0, 1): 範囲 [0, 1) の一様乱数
    // T_t: 時刻 t における温度（単調減少）
    // Pr: 受理確率
2   x ← 任意の状態;
3   for t = 0 to ∞ do
4       x' ← x の近傍 Nb(x) から選んだ任意の状態;
5       ΔE ← E(x') − E(x);
6       ω ← random(0, 1);
7       if ω ≤ Pr(T_t, ΔE) then x ← x' ;
```

Algorithm 2: Simulated Annealing

体に近い大きいサイズの再標本に対して最適化を行うので局所探索の振る舞いを行うことになる．また，試行の度に再標本の取り換えを行う．この手法は高温時には小さいサイズの再標本に対してのみ評価を行うため，従来の SA より高速に焼きなますことが可能である．Algorithm 3 に AIR の概略を示す．

```
1 procedure AIR
    // size(t): 時刻 t における再標本サイズ（単調増加）
2   x ← 任意の状態;
3   for t = 0 to ∞ do
4       x' ← x の近傍 Nb(x) から選んだ任意の状態;
5       S' ← ランダムに選んだ再標本，ただし S' ⊆ S かつ |S'| = size(t);
6       if E(x', S') − E(x, S') ≤ 0 then x ← x' ;
```

Algorithm 3: Annealing by Increasing Resampling

本研究では，m 次元に特徴抽出した各次元 0 から 255 の座標値を持つデータを用い，幅 w が 16 ビットや 32 ビットのスケッチを作成することを考える．データベース内から

ピボットの中心点候補をランダムに w 個選び，焼きなましを開始する．ピボットを一つ選んで，そのピボットの m 次元から一つの次元を選び，その座標値を最小から最大まで変化させてスコアが最良のもの，すなわち衝突確率が最小のものを選ぶ．最良のものが複数あるときは，ランダムに選ぶようにした．スコア計算では，中心点の一部の値しか変化しないことを利用して，256 の候補のスコアを効率よく計算できるように工夫した（計算途中結果の再利用，部分スコアの利用など）．これにより，より多くの候補から遷移先を選択できるようにして，焼きなまし法がより効果的に動作するようになった．とくに，従来の SA の高温時に相当する小さいサイズの再標本の評価を用いるランダムウォークで多くの点を走査できるようになったと考えている．

3.3 最適化手法の比較実験

約 690 万件 64 次元の画像データを用いて，$w = 32$ ビットのスケッチの最適化を三つの手法 BP，QBP，AIR で行った．QBP は，Algorithm 1 で示したもので，BP は，QBP における 2 値量子化を省いたものである．最適化の目的関数は衝突確率を用いた．評価用の標本 S は，データベースからランダムに選んだ 10,000 点を用いた．比較のため，ピボット探索にかかる時間が同程度 (100 秒程度）となるように以下のように設定する．

- BP，QBP： 各次元試行回数 $Trials = 1,000$ 回
- AIR： 初期サンプル 100，総試行回数 10,000 回，局所探索 2,028 回

AIR における局所探索の回数は各ピボットの各次元 1 回ずつ変更を行える程度の 2,028

回 (ピボット数 × ピボットの特徴次元数) にしている．AIR は，初期状態に依存しないため，$Trials = 1$ の BP で得られるものを初期状態として実験を行う．

表 3.1　AIR の適用

最適化	衝突確率 $[\times 10^{-7}]$	精度 [%]
BP	25.9 ± 6.20	88.5 ± 1.05
QBP	22.0 ± 4.95	90.9 ± 0.620
AIR	10.2 ± 4.33	87.8 ± 0.855

AIR を適用することにより，従来の最適化手法よりも上手く最適化できていることがわかる．しかし検索時の精度はよくなっていない．最小衝突法で最適化することで精度が向上する傾向があることがわかっているが，最適化しすぎると衝突してもよい（距離が近い）データまで分離してしまうため，精度が下がると考えられる．そのため，評価関数の見直しが必要である．

第 4 章

次元縮小の量子化像としてのスケッチ

従来，スケッチは，局所性鋭敏ハッシュ（LSH）の一種として扱われてきた．本論文で用いる球面分割によるスケッチは，次元縮小 Simple-Map の量子化像としてみなすことができる．このことから，質問とデータ間の距離下限を質問とデータのスケッチを用いて求めることが可能であることがわかる．この距離下限値に基づく優先順位付けにより，ハミング距離よりも高精度・高速な類似検索が可能となる．本章は，論文 [7] に基づいている．

4.1 質問とスケッチ間の距離下限値に基づく優先順位付け

本研究では，第 1 段階の検索において，距離下限値に基づく優先順位付け [7] を用いる．ピボット集合を $P = \{(p_0, r_0), ..., (p_{w-1}, r_{w-1})\}$ とする．質問 q と $BP_{(p_i,r_i)}$ の分割境界

との最小距離を $e_i(q,P)$ とすると，

$$e_i(q,P) = |D(p_i,q) - r_i|$$

質問とデータのスケッチが異なるとき，少なくとも質問と分割境界との最小距離だけ離れていることは明らかである．これを用いて， つぎのように，$D(q,x)$ の距離下限値 $b_i(q,s_P(x))$ を求めることができる．

$$b_i(q,s_P(x)) = \begin{cases} 0, & \text{if } BP_{(p_i,r_i)}(q) = BP_{(p_i,r_i)}(x), \\ e_i(q,P), & \text{otherwise.} \end{cases}$$

ここで，$b_i(q,s_P(x))$ を求めるために，x については，そのスケッチのみがあればよく，x 自身は不要であることに注意されたい．第 1 段階における解候補選択の基準として距離下限値 $b_i(q,s_P(x))$ に基づいて優先順位付けする．下限値の最大である以下の $socre_\infty$ を優先順位付けとするとき，真の距離下限値であるので安全な枝刈りが可能である．

$$score_\infty(q,s_P(x)) = \max_{i=0}^{w-1} b_i(q,s_P(x))$$

また，距離下限値の総和である以下の $score_1$ は，距離の下限となることが保証できないが，これを用いるとより高精度の検索が行える．

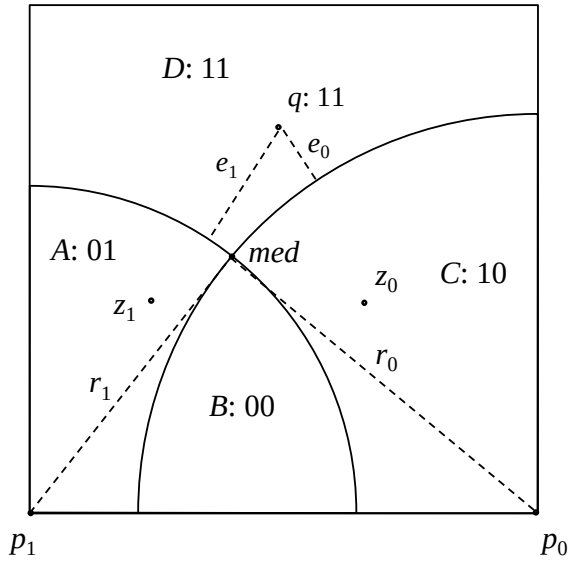
$$score_1(q,s_P(x)) = \sum_{i=0}^{w-1} b_i(q,s_P(x))$$

スケッチ間の距離の場合は，2 値に量子化しているので，「一致」か「不一致」の 2 者択一になり，量子化誤差は大きい．提案手法では，質問側をスケッチではなく元のデータを用いて，量子化誤差として消えてしまう距離情報を部分的に復元している．これがハミング

距離よりも $score_\infty$ が高精度の優先順位付けになる理由だと考えられる．ただし，$score_1$ が距離下限値でないにも関わらず $score_\infty$ よりも高精度の優先順位付けになる理由は判明していない．

4.2 QBP によるスケッチと優先順位の例

図 4.1 に，2 次元ユークリッド平面上における QBP による 2 ビットスケッチと優先順位の例を示す．データの中央値を med とする．データからランダムに選んだ 2 点を z_0, z_1 とすると，med を閾値として 2 値量子化すると，それぞれ p_0, p_1 になる．量子化した点と中央値の距離を r_0, r_1 として，2 個のピボット集合 $P = \{(p_0, r_0), (p_1, r_1)\}$ を用いると，2 ビットスケッチが得られる．中央値との距離を半径とするのは，球面分割によってデータがほぼ半々に分けられるからである．2 個の球面分割によって，平面は，4 個の部分空間 A, B, C, D に分けられる．A または B, C, D の任意の点 x に対して，$s_P(x)$ は，それぞれ 01 または 00, 10, 11 である．図のように，q を両方の球の外側の質問とすると，q からの部分空間の任意の点に対する優先順位は，図の右の表のようになる．ただし，e_0, e_1 は，q と分割境界との最小距離である．ここで，従来の優先順位付けであるハミング距離では部分空間 A と C が区別できないが，$score_1$ を用いるとすべてが分離できることに注意しよう．また，質問 q を部分空間 D にとる場合でも，e_0, e_1 の大小によって，優先順位が変化することがあることにも注意されたい．



部分空間	優先順位付け		
	ハミング	$score_\infty$	$score_1$
A	1	e_1	e_1
B	2	e_1	e_0+e_1
C	1	e_0	e_0
D	0	0	0

質問 q からの優先順位

$D < A = C < B$（ハミング距離）

$D < C < A = B$（$score_\infty$）

$D < C < A < B$（$score_1$）

図 4.1　QBP による 2 ビットスケッチと優先順位付け

4.3　優先順位付けの比較実験

優先順位付けとして，ハミング距離と $score_\infty$，$score_1$ を比較した 32 ビットスケッチを用いた類似検索の実験結果を紹介する [7]．画像データおよび音楽データを用いた検索精度の結果を図 4.2 と図 4.3 に示す．ここでは，32 ビットのスケッチを用いているので，第 1 段階検索は，質問とデータベースの全スケッチを照合して行っている．解候補数をデータ件数の 0.01%〜1.0% のときの優先順位付けを用いた検索精度を示す．距離下限値に基づくスコアによる優先順位付けを用いることでハミング距離を用いる場合よりも検索精度が向上していることがわかる．特に $score_1$ が最高精度を達成している．

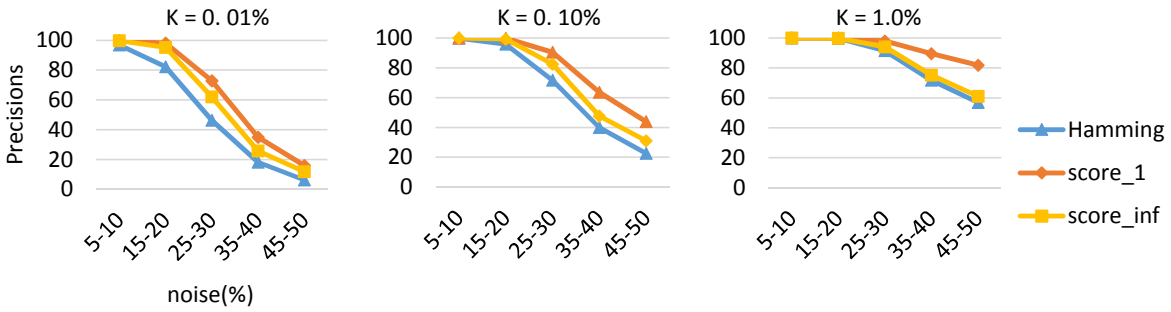


図 4.2　優先順位付けを用いた検索精度（画像データ）

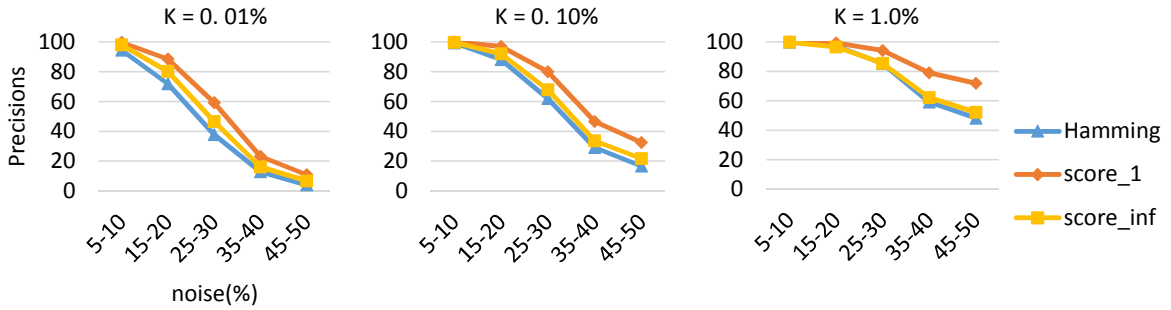


図 4.3　優先順位付けを用いた検索精度（音楽データ）

第 5 章

ビット幅の狭いスケッチを用いる高速検索

本章では，従来よりもビット幅が狭いスケッチを用いた検索について議論する．従来は 32 ビット以上の幅のスケッチを用いていた．本章では，まず，データベースのデータ数を数百万であると仮定し，スケッチの幅を 16 ビットに固定する．実際には別の幅を用いる方が効率が良くなる可能性があるが，これについては，実験の章で議論することにする．本章は，論文 [9, 10, 8] に基づいている．

16 ビットスケッチの総数は，2^{16}=約 6 万 6 千であるので，スケッチそのものをキーとするバケット法でデータを管理することが可能である．また，数百万個の個々のデータのスケッチとの照合を行うよりも，すべての 16 ビットスケッチとの照合を行えば十分であり，そのコストはデータベースサイズに依らない．各スケッチに射影されるデータ数は，

ある程度均等であると考えられるので，実際の検索時には，質問のスケッチと類似するごく一部のスケッチしか必要ない．したがって，質問のスケッチに近い順にスケッチを列挙するアルゴリズムを利用することにより，バケット法でデータを管理していれば，スケッチ間の照合が不要となり，第 1 段階検索はほぼコストを無視できるように高速化が可能である．

5.1 バケット法によるデータ管理

従来の第 1 段階検索では質問のスケッチとデータベース内のすべてのデータのスケッチとの距離を計算し，距離が近いデータ K 個を解候補としていた．しかし，ビット幅の狭いスケッチの総数はデータベース数に対して小さいので，すべてのスケッチパターンと照合する方が計算コストが小さくなる．そこでバケットを用いた検索を提案する．あらかじめデータベースをスケッチ順にソートしておく．スケッチをキーとするバケットを用意し，そのスケッチとなるデータのソートされたデータベースにおける先頭位置と個数を返すようにすることで，1 つのスケッチの照合で衝突しているデータをすべてを得ることができる．すべてのスケッチパターンと距離計算し，距離が近い順にスケッチをキーとしてバケットに与えることで，解候補 K 個のデータを取り出すことができる．

ここで，すべてのスケッチパターンと距離計算をしているが，スケッチを距離が近い順に列挙することができればこの距離計算は不必要になり，さらに高速化することが可能である．

5.2 ハミング距離順の列挙を用いる検索の高速化

16 ビットのスケッチを用いる検索は，ハミング距離を用いる場合は比較的容易に高速化することができる．事前にすべての 16 ビットパターンを ON ビットの個数の昇順にソートしたものを準備しておく．質問のスケッチとこのビットパターンとの XOR を求めると距離の順にスケッチを求めることができるので，これを用いて順次第 2 段階検索を実行する．これにより，事実上，第 1 段階の検索コストを無視できるようになる．

Algorithm 4 に 16 ビットスケッチのハミング距離を用いた最近傍検索アルゴリズムを示す．ここに，データベースは，以下のように，スケッチをキーとするバケットを用いて管理すると仮定している．

- $x[0], x[1], \ldots, x[n-1]$: 特徴データの配列，ただし，データがメモリ上でスケッチ順になるようにソートしておく．（ポインタを介した間接的なソートではない）
- $id[i]$: 特徴データ $x[i]$ のデータ ID.
- $\mathit{first}[s]$: スケッチが s であるデータの配列 x における先頭位置.
- $num[s]$: スケッチが s であるデータ数.

このようにすると，スケッチが s であるデータは，

$$x[\mathit{first}[s]], x[\mathit{first}[s]+1], \ldots, x[\mathit{first}[s]+num[s]-1]$$

になる．また，ハミング距離による検索のための準備として，すべての 16 ビットパター

```
// x[0], ..., x[n − 1]: スケッチ順にソートした特徴データの配列
// id[i]: 特徴データ x[i] のデータ ID
// first[s]: スケッチが s であるデータの配列 x における先頭位置
// num[s]: スケッチが s であるデータ数
// K: 第 1 段階で得る候補数＝実距離計算回数
// mk[0], ..., mk[2^w − 1]: ビットパターンを ON ビット数の昇順に並べた配列
// w: スケッチの幅（ビット数）
1  function SEARCH(query, s, NN, nearest, checked)
2      for i = first[s] to first[s] + num[s] − 1 do
3          checked ← checked + 1;
4          if D(query, x[i]) ≤ nearest then
5              (NN, nearest) ← (id[i], D(query, x[i]));
6          if checked ≥ K then
7              return(NN, nearest, checked);
8      return(NN, nearest, checked);
9  function NNSEARCHBYHAMMING(query)
10     (NN, nearest, checked) ← ("none", ∞, 0);
11     for i = 0 to 2^w − 1 do
12         s ← sketch(query) ⊕ mk[i];
13         (NN, nearest, checked) ←SEARCH(query, s, NN, nearest, checked);
14         if checked ≥ K then
15             return NN;
16     return NN;
```

Algorithm 4: ハミング距離順の列挙による最近傍検索アルゴリズム

ンを ON ビット数の昇順に並べたマスク配列 mk を用意しておく．

$$mk[0], \ldots, mk[2^{16} - 1]$$

5.3 優先順位付け $score_\infty$ を用いる検索の高速化

距離下限値に基づく $score_\infty$ を優先順位付けとして用いる場合は，距離下限値が質問毎に異なるため，ハミング距離を利用する場合と異なり，事前に準備したビットパターン列との XOR でスケッチを列挙することはできない．しかし，以下に説明するように，グレイコード生成法を利用したアルゴリズムにより，質問のスケッチとの $score_\infty$ が小さい順にスケッチを列挙することができる．

まず，3 ビットスケッチのときの具体例を示す．ここでは，以下を仮定する．

- ビット列を 2 進数と見たときの 2^i の位を位置 i とする（$i = 0, 1, 2$）．
- (p_i, r_i) : 位置 i に対応する球面分割のピボット．
- $P = \{(p_0, r_0), (p_1, r_1), (p_2, r_2)\}$: ピボット集合．
- q: 質問．
- $e_i = |D(q, p_i) - r_i|$: 質問と位置 i に対応する球面分割境界の最小距離．簡単のため，これらの距離下限値は，つぎを満たしているとする．

$$e_2 \geq e_1 \geq e_0$$

そうすると，質問とスケッチ間の $score_\infty$ は，小さい順に 0, e_0, e_1, または e_2 の高々 4 種類しかない．質問のスケッチを $s_P(q) = 011$ とすると，それぞれの $score_\infty$ をもつスケッチは，以下のようになる．

$score_\infty = 0$ のスケッチ

011 自身.

$score_\infty = e_0$ のスケッチ

011 と位置 0 のビットだけ異なるもの，つまり，010．これは $s_P(q)$ と 001 の XOR で求められる.

$score_\infty = e_1$ のスケッチ

011 と位置 2 のビットは同じで，位置 1 のビットが異なり，位置 0 のビットは任意．つまり，000 と 001．これらは，$s_P(q)$ とそれぞれ 010, 011 の XOR である.

$score_\infty = e_2$ のスケッチ

011 と位置 2 のビットが異なり，残りは任意．つまり，100, 101, 110, 111．これらは，$s_P(q)$ とそれぞれ 100, 101, 110, 111 の XOR.

このように，$s_P(q)$ との $score_\infty$ が小さい順のスケッチは，$s_P(q)$ と 2 進数で小さい順となるつぎのビットパターン列との XOR で求められる.

$$000, 001, 010, 011, 100, 101, 110, 111$$

$score_\infty$ はこのビットパターンの先頭の ON ビットの位置で決まることに注意すれば，これをグレイコード順に並べてもよいことがわかる.

$$000, 001, 011, 010, 110, 111, 101, 100$$

これは，2 進数の値の順序とグレイコードの順序は，どちらも先頭の ON ビットが立って

表 5.1　グレイコード生成とスケッチの列挙

000	011	0
$000 \oplus 001 = 001$	$011 \oplus 001 = 010$	e_0
$001 \oplus 010 = 011$	$010 \oplus 010 = 000$	e_1
$011 \oplus 001 = 010$	$000 \oplus 001 = 001$	e_1
$010 \oplus 100 = 110$	$001 \oplus 100 = 101$	e_2
$110 \oplus 001 = 111$	$101 \oplus 001 = 100$	e_2
$111 \oplus 010 = 101$	$100 \oplus 010 = 110$	e_2
$101 \oplus 001 = 100$	$110 \oplus 001 = 111$	e_2

いる位置の昇順であるからである．

どのビット位置を反転させるかの系列は, 表 5.1 のように，0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0 のようになっている．このグレイコードの性質を利用すると，効率的な列挙が可能となる．質問に対する各ビット位置の距離下限値の順序が決まれば，それを用いた相対的な位置のビット反転操作を行うのである．反転させるビット位置は $bitcount(i \oplus (i+1)) - 1$ で求めることが可能である．オール 0 のビットパターンから始めず，表 5.1 のように，質問のスケッチ $s_P(q)$ から始めることで質問からのスコアが小さい順にスケッチを列挙することが可能である．グレイコードの性質を利用することによりスケッチを列挙するアルゴリズムは非常に単純になる．なぜならば，その並びのつぎのスケッチを得るための操作は，ちょうど *1* ビットの反転操作で行えるからである．このことにより，定数遅延の列挙が可能となっている点に注意しよう．

ここで，上に述べた手法で正しくスケッチを列挙できる理由を説明しておく．整数 i に対応するグレイコードを $g(i)$ とする（$i = 0, 1, \ldots$）．つまり，$g(0) = 000$, $g(1) =$

$001, \ldots, g(7) = 100$ である．そうすると，生成すべき列挙の第 i 番目のスケッチは，$s_P(q) \oplus g(i)$ である．排他的論理和 $\oplus$ の性質により，つぎが成り立つ．

$$(s_P(q) \oplus g(i)) \oplus (s_P(q) \oplus g(i+1)) = g(i) \oplus g(i+1)$$

ここで，グレイコードの性質より，$g(i) \oplus g(i+1)$ は ON ビットが 1 個だけのビットパターンであることに注意しよう．

$$g(i) \oplus g(i+1) = (1 \ll (bitcount(i \oplus (i+1)) - 1))$$

であるので，$s_P(q)$ から始めて，グレイコードを生成するときと同じビット反転操作を適用すると，求めるスケッチが列挙できる．ここで，$\ll$ は左論理シフト演算子である．

質問と分割境界との最小距離は，$e_2 \geq e_1 \geq e_0$ を満たしているとは限らないので，実際には，これらの順位を求めておく．Algorithm 5 に示したアルゴリズムでは，$bidx[0], \ldots, bidx[w-1]$ は，$0, 1, \ldots, w-1$ の並べ替えで，つぎを満たしているとする．

$$e_{bidx[w-1]} \geq \cdots \geq e_{bidx[1]} \geq e_{bidx[0]}$$

列挙アルゴリズムは，この順位だけで最小距離そのものは用いていない．また，関数 Search はハミング距離を用いるアルゴリズム（Algorithm 4）で示している．

5.4 優先順位付け $score_1$ を用いる検索の高速化

優先順位付けには $score_\infty$ よりも $score_1$ を用いる方が精度がよくなることがわかっている．しかし，$score_\infty$ は距離パターンが高々 $w+1 = 17$ 個しかないことを利用して高

```
// w : スケッチの幅（ビット数）
// K : 第 1 段階における候補数 = 実距離計算回数
function SearchByScoreInf(query)
    query に対する距離下限値の順位表 bidx[0], ..., bidx[w − 1] を準備する;
    (NN, nearest, checked) ← ("none", ∞, 0);
    s ← sketch(query);
    (NN, nearest, checked) ←Search(query, s, NN, nearest, checked);
    if checked ≥ K then
        return NN;
    for i = 0 to 2^w − 1 do
        s ← s ⊕ (1 ≪ bidx[bitcount(i ⊕ (i + 1)) − 1]);
        (NN, nearest, checked) ←Search(query, s, NN, nearest, checked);
        if checked ≥ K then
            return NN;
    return NN;
```

Algorithm 5: 優先順位付け $score_\infty$ 順の列挙による最近傍検索アルゴリズム

速化しているが，$score_1$ の値は最大 $2^w = 2^{16}$ 個あるため同様の高速化はできない．以下に示すように，優先度付き待ち行列を利用した効率的な列挙が可能である．

アルゴリズム (Algorithm6) は，質問との $score_1$ が小さい順にスケッチを列挙できることを利用した最近傍検索手法である．アルゴリズム (Algorithm5) と同様に，各質問に対して，まず，距離下限値とその順位表を求めておく．このアルゴリズムで質問との $score_1$ の昇順にスケッチが正しく列挙できていることを以下で説明する．簡単のために，距離下限値は，

$$e[w-1] \geq \cdots \geq e[1] \geq e[0]$$

を満たしていると仮定する．実際には，アルゴリズム (Algorithm5) と同様に，距離下限値の順位表を用いて，相対ビット位置で操作を行っている．

スケッチの列挙には優先度付き待ち行列 PQ を用いている．PQ の要素は，$(score, s, i)$ の組で，s はスケッチ，$score = score_1(query, s)$，$MSB(qs \oplus s) = i - 1$ である．ここで，MSB は最左ビット位置，qs は $query$ のスケッチとする．

まず，ループの外側では，質問のスケッチ qs と一致するもの，つまり $score_1 = 0$ であるものを検索している（6 行目）．続いて，$(e[0], qs \oplus 1, 1)$ を PQ に挿入している（9 行目）．このとき，$MSB(qs \oplus (qs \oplus 1)) = MSB(1) = 0$ であり，$score_1(query, qs \oplus 1) = e[0]$ であることに注意しよう．

ループ内では，PQ からスコアが最小の $(score, s, i)$ ものを取り出して（11 行目），スケッチが s であるものを検索し（12 行目），その後，2 個の要素

$(score - e[i-1] + e[i], (s \oplus (1 \ll i)) \oplus (1 \ll i-1), i+1)$ と $(score + e[i], s \oplus (1 \ll i), i+1)$

を PQ に挿入している（17 行目及び 18 行目）．ここで，PQ から取り出されたスケッチ s と質問のスケッチ qs に対して，

$$s \oplus qs = 1X，\text{ただし，} X \text{ は } i-1 \text{ 桁の 2 進数}$$

となっていて，PQ に挿入されるスケッチは，$10X$ と $11X$ となっていることに注意しよう．このことから，アルゴリズムでは，幅 w ビットのすべてのスケッチを重複なしで列挙する手順を用いていることがわかる．また，列挙されるスケッチのスコアが $score_1$ の昇順になっていることは，PQ から取り出されたスケッチのスコアはその後で挿入されるス

```
// w : スケッチの幅（ビット数）
// K : 第 1 段階における候補数 = 実距離計算回数
// PQ : 優先度付き待ち行列（要素は（score, sketch, i）で score が優先度）
1  function SearchByScore1(query)
2      qurey に対する距離下限値 e[0], ..., e[w − 1] を求める;
3      query に対する距離下限値の順位表 bidx[0], ..., bidx[w − 1] を準備する;
4      (NN, nearest, checked) ← ("none", ∞, 0);
5      s ← sketch(query);
6      (NN, nearest, checked) ←Search(query, s, NN, nearest, checked);
7      if checked ≥ K then
8          return NN;
9      PQ ← (e[bidx[0]], s ⊕ 1 ⟪ bidx[0], 1);
10     while PQ is not empty do
11         (score, s, i) ← PQ;
12         (NN, nearest, checked) ←Search(query, s, NN, nearest, checked);
13         if checked ≥ K then
14             return NN;
15         if i < w then
16             s ← s ⊕ (1 ⟪ bidx[i]);
17             PQ ← (score − e[bidx[i] − 1] + e[bidx[i]], s ⊕ (1 ⟪ bidx[i] − 1), i + 1);
18             PQ ← (score + e[bidx[i]], s, i + 1);
```

Algorithm 6: 優先順位付け $score_1$ 順の列挙による最近傍検索アルゴリズム

ケッチのスコアより大きくないことから明らかである.

第 6 章

実験

本章では，2.7 節で説明した画像や音楽データを用いた実験結果を示す．スケッチの幅は，従来手法では 32 ビットのものを用いる．ここでは，の実験結果についてまとめる．まず，6.1 節では，論文 [10, 9] で報告した提案手法の 16 ビットスケッチを用いた実験結果を示す．これまでは，提案手法のスケッチの幅は 16 ビットに固定していた．実際には，これ以外のビット幅を用いる方が効率が良い可能性がある．6.2 節では，データベースのデータ数や特徴次元数と最適なスケッチ幅の関係について，論文 [8] で報告した 13〜28 ビットのスケッチを用いた実験結果に基いて議論する．いずれの場合も，3.1 節の Algorithm 1 で示した SelectPivotQBP を用いて衝突が少なくなるように選んだピボット集合を用いている．

実験に用いた PC の諸元を表 6.1 に示す．

表 6.1　実験環境

CPU	Intel(R) Xeon(R) CPU E5-2640 2.5 GHz
memory	64GBytes

表 6.2　16 ビットスケッチのバケット

データベース	画像データ	音楽データ
平均要素数	105	108
空バケット数	908 (1.5%)	2104 (3.1%)
要素数 10 以上のバケット数の割合（%）	87%	74%

6.1　16 ビットスケッチによる検索の精度と速度

表 6.2 に，それぞれのデータに対して，16 ビットスケッチをキーとするバケットの様子（平均要素数（= データ数/2^{16}），空バケット数，要素数 10 以上のバケット数の割合）を示した．多くのバケットは表 6.2 に示されるように 10 個以上の要素を有するので，スケッチの順番でデータをソートすることによる第 2 段階の高速化が期待できる． ソートの有無による比較実験では，約 3 倍の高速化が確認できている．

画像データおよび音データに対する実験結果をそれぞれ表 6.3，表 6.4 に示す．検索精度とは，得られた解が実際の最近傍解の実距離と一致している確率である．ここで，score は検索優先順位付け，width はスケッチの幅（ビット数），K は第 1 段階での候補数（データ件数に対する割合（%）），sketches は検索時に列挙されたスケッチ数（16 ビットのみ，6 万 6 千個に対する割合（%），100% は列挙を用いない場合），time は 1 質問当たりの検

索時間（ミリ秒）を表す（1st st. は第 1 段階の検索時間）．列挙を用いる場合は，第 1 段階検索のコストは分離することは困難なので省略している．実距離計算回数 K は検索精度が同程度（画像では 70%，音データでは 65%）以上になるように 32 ビットでは 0.1%，16 ビットでは 1.0% とする．All は全質問に対する平均精度を表す．32 ビットスケッチに対する $K = 0.1\%$ の設定は，筆者らの研究室における R-Tree による検索速度（100 ミリ秒／質問）より高速である程度の精度（70% 程度）の検索を達成するものである．

また，比較のために，画像データに対する全探索を実行したところ，1 質問当たり 550 ミリ秒の検索時間を要することがわかった．最近傍探索時には，質問とデータ間の距離計算をそれまでに見つかった暫定解との距離を超える段階で打ち切ることで高速化することができ，探索時間を 280 ミリ秒まで短縮できる．したがって，精度 70% 程度であれば，従来法の 32 ビットスケッチでハミング距離を用いた検索（29.8 ミリ秒）でも，全探索より 10 倍あるいは 20 倍程度高速であることがわかる．さらに，提案手法の 16 ビットスケッチを用いると，優先順位付けに $score_\infty$ を用いた検索（2.68 ミリ秒）は，全探索より 100 倍以上あるいは 200 倍以上高速であり，しかも高精度（79.7%）であることがわかる．

これらの表から，ノイズレベルが高くなるにつれて，つまり，最近傍解が遠くなるにつれて，検索精度が低くなっていることがわかる．これは，高次元データにおける「次元の呪い」の影響と考えられる．優先順位付けにハミング距離を用いる場合は，いずれのデータベースに対しても，16 ビットスケッチ（$K = 1.0\%$）で 32 ビットスケッチ（$K = 0.1\%$）と同等の検索精度を達成できる．優先順位付けに $score_\infty$ や $score_1$ を用いると，検索精度が向上している．16 ビットスケッチの列挙による高速化の有効性も確認できる．列挙

表 6.3　画像データに対する検索時間と検索精度

score	ハミング距離			$score_\infty$			$score_1$		
width	32	16		32	16		32	16	
$K(\%)$	0.1	1.0		0.1	1.0		0.1	1.0	
sketches(%)	–	100	0.76	–	100	0.73	–	100	0.62
1st st. time	28.7	4.36	–	35.6	3.23	–	32.0	4.90	–
total time	29.8	7.16	2.85	36.9	6.06	2.68	33.2	7.76	3.07
All	70.2	73.4	73.0	74.3	79.7	79.7	80.2	85.1	85.6
very-near	99.8	99.7	99.6	100	100	100	100	100	100
near	96.9	94.4	94.8	99.1	99.3	99.3	100	100	99.8
middle	80.4	80.3	79.3	85.7	88.8	88.8	92.7	94.4	94.4
far	46.0	53.9	53.3	52.4	63.5	63.5	63.7	73.2	74.1
very-far	28.0	38.5	37.9	34.4	47.2	47.2	44.7	58.0	59.7

を用いるかどうかで精度に若干の差が出ているが，これは同一の優先順位のものがある場合に手法によって選択されるものが異なるからである．また，検索時に列挙される 16 ビットスケッチは，ごくわずかであることがわかる．列挙による高速化を用いると，検索速度は，従来の 32 ビットスケッチを用いる場合より 10 倍程度高速化できている．

従来の手法では，R-Tree など他の手法の検索速度より高速にするためには，K を大きくして精度を高くすることができなかったが，提案手法は，より高い精度が要求される場合であっても，高速な検索が期待できる．精度が 90% を越えるより大きな K で比較した結果を表 6.5，表 6.6 に示す．精度を 90% 以上に保つためには，画像データにおいては，従来の 32 ビットスケッチを用いた検索速度（1 質問当たり 139 ミリ秒，$score_\infty$ を用いた 106 ミリ秒，$score_1$ を用いて 107 ミリ秒）は，素朴な全探索（550 ミリ秒）より 4 倍

表 6.4　音データに対する検索精度

score	ハミング距離			$score_\infty$			$score_1$		
width	32	16		32	16		32	16	
K(%)	0.1	1.0		0.1	1.0		0.1	1.0	
sketches(%)	–	100	0.55	–	100	0.48	–	100	0.46
1st st. time	30.3	4.33	–	36.3	3.23	–	34.4	4.63	–
total time	31.7	7.89	3.58	38.0	6.82	3.38	36.0	8.30	3.76
All	65.5	66.8	66.0	63.6	75.1	73.4	72.3	77.8	77.7
very-near	99.7	98.7	98.2	99.5	99.8	99.7	100	100	100
near	93.3	89.1	89.3	92.2	95.3	94.8	97.2	97.2	97.4
middle	70.6	68.2	67.6	67.4	80.3	78.5	79.8	83.0	83.1
far	40.2	44.2	43.7	36.9	55.6	53.1	51.2	61.5	61.2
very-far	23.9	33.7	31.4	21.9	44.5	41.2	33.2	47.4	46.9

表 6.5　画像データに対する検索精度 90% 以上の実験結果

score	ハミング距離			$score_\infty$			$score_1$		
width	32	16		32	16		32	16	
K(%)	2.0	6.5		1.5	5.0		1.0	2.5	
sketches(%)	–	100	5.8	–	100	4.1	–	100	1.7
time(ms)	139	22.0	17.5	106	16.8	12.8	107	12.0	7.32
All	91.5	90.2	90.2	90.1	92.0	90.6	93.8	91.4	92.0
very-near	100	100	100	100	100	100	100	100	100
near	100	99.7	99.7	99.9	100	99.9	100	100	100
middle	97.4	95.4	95.4	97.3	97.3	96.6	99.1	97.3	97.7
far	84.4	82.4	82.4	81.8	85.8	83.4	88.7	85.3	86.0
very-far	76.0	73.8	73.8	71.6	77.0	73.4	81.5	74.7	76.4

表 6.6　音データに対する検索精度 90% 以上の実験結果

score	ハミング距離			$score_\infty$			$score_1$		
width	32	16		32	16		32	16	
K(%)	3.5	10		3.5	8.0		1.5	4.0	
sketches(%)	−	100	7.4	−	100	5.5	−	100	2.2
time(ms)	227	37.4	32.7	188	30.1	25.1	160	18.7	14.5
All	90.8	90.4	90.1	90.3	92.2	90.6	92.0	90.9	90.6
very-near	100	100	100	100	100	99.8	100	100	100
near	99.2	98.8	98.6	98.6	99.1	98.9	99.4	99.2	99.5
middle	93.9	92.9	92.5	92.9	94.7	93.4	95.2	94.1	94.0
far	85.2	84.2	83.5	84.3	86.8	85.7	86.9	84.6	84.1
very-far	76.0	76.3	76.2	75.8	80.5	75.9	78.6	76.6	75.4

程度，あるいは距離計算の打ち切りを行う全探索（280 ミリ秒）より 2 倍程度の高速化にしかならない．16 ビットスケッチでは 32 ビットのときより，K は大きくする必要があるが，8 倍から 10 倍高速化でき，画像データに対しては，全探索より 40 倍または 20 倍程度の高速化が達成できる．また，$score_1$ を用いると K をあまり大きくしなくても高精度を達成でき，列挙を用いることで最高性能を達成している．

ここで，32 ビットスケッチを用いる従来手法と提案手法の比較および優先順位付け（ハミング距離，$score_\infty$，$score_1$）の効果，スケッチの列挙による高速化の確認のために，第 1 段階の検索候補数 K を変化させて，検索時間と精度を測定した．図 6.1 は，Database 1 を使用した五つ検索方法の結果を示している．横軸は質問に関する平均時間（ミリ秒(ms)）であり，縦軸は検索精度である．従来方法の 32 ビットスケッチを用いる場合は，優先順位付けは最も良い $score_1$ を用いた結果のみを示す．その他は，16 ビットスケッチ

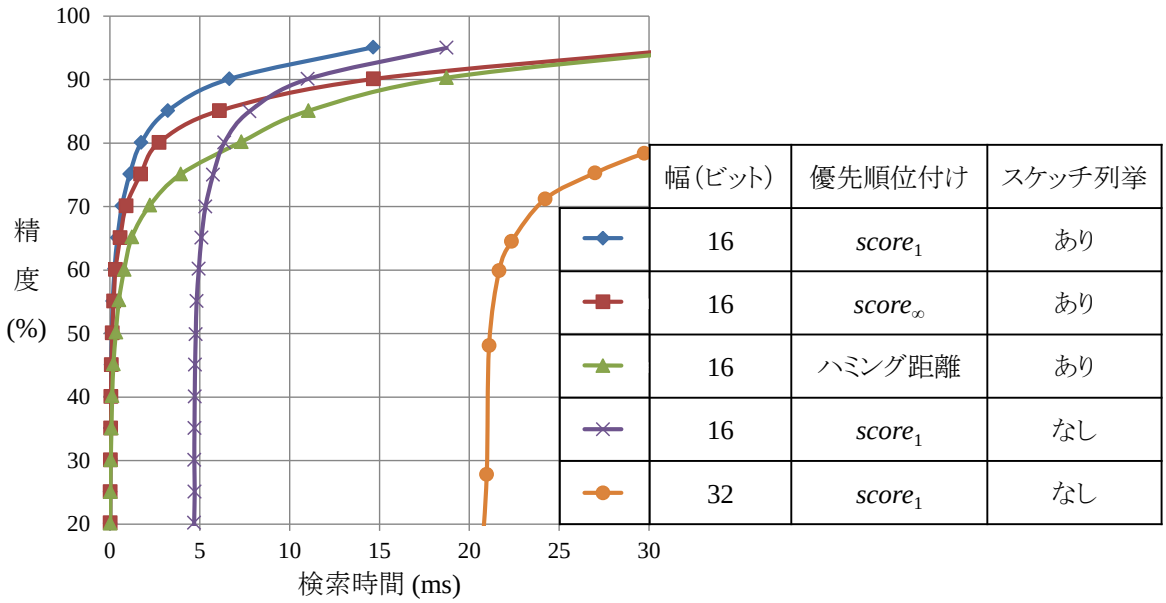


図 6.1　検索時間と精度

を使用する方法である．ハミング距離と $score_\infty$ を用いた方法はスケッチの列挙による高速化を行った結果を示す．$score_1$ を使用した 16 ビットスケッチの場合，スケッチの列挙による高速化ありとなしの二つの方法を示す．正確な検索にはより大きな K が必要になるため，どの方法でもより高い精度を得るためにより多くの時間が必要である．従来の方法の遅さの主な理由は，第一段階での全探索に起因しており，少なくとも約 21ms を必要とすることがこの図からも読み取れる．

$score_1$ のスケッチの列挙は予想どおり非常に効率的に機能し，90 %の精度で約 6 ミリ秒の検索時間は，全検索の 550 ミリ秒より 100 倍近く速いことがわかる．16 ビットの 2^{16} 個のスケッチすべてと質問のスケッチをマッチングするためには，約 5 ミリ秒の時間が消費されることが，列挙による高速化ありとなしの時間差に現れている．

6.2 最適なスケッチのビット幅

6.2.1 画像データにおける最適ビット幅

各データベースおよび各優先度について，最初に検索精度が 90 %になる候補数を決定した．各データベースごとの実験結果を表 6.9，表 6.10，表 6.11，表 6.12，表 6.13 に示している．また，それぞれの実験結果は次の項目別にまとめている．すべての値は質問ごとの平均である．

- K: 候補数 (データ件数に対する割合).
- *sketches*: 列挙されたスケッチ数.
- *empty*: 列挙されたスケッチのうち空のバケット数.
- *elements*: 列挙された空でないバケットの平均要素数.
- *time*: 検索時間 (ms)
- *unsorted*: データベースのソートなしでの検索時間 (ms).
- *enum*: スケッチの列挙にかかる時間 (ms).
- *sorting effect*: 列挙にかかる時間を除いたソートによる高速化率.

表 6.9 からわかるように，優先順位付けにハミング距離，$score_\infty$，$score_1$ のどれを用いても，最適なスケッチの幅は 16 ビットよりも長くなる．一般に，スケッチの幅が広いほど，90% の精度を達成するのに必要な候補数は小さくなる．通常，第 2 段階の検索の計算コストは K によって決定されるが，スケッチの幅が広くても検索時間が短くなるとは

限らない．この現象は，データの並べ替えによるメモリアクセスの局所性の改善によるものと考えられる．これは，ソートされたデータベースを使用しない検索の最適な幅が長くなるという事実によっても確認できる．表のソート効果行に示すように，より広いスケッチを使用すると，データベースのソートによる効果は小さくなる．したがって，最適な幅は，スケッチの選択性とメモリの局所性の向上との間のトレードオフによって決定されていると考えられる．

6.2.2 最適ビット幅とデータベースのデータ件数と特徴次元数

表 6.7 に各画像データベースのスケッチの最適ビット幅，表 6.8 に $score_1$ の列挙を用いた検索の高速化率を示している．Database 1、Database 4、および Database 5 の結果を比較することで，データ件数による影響を考える．これらのデータは、同じ特徴次元数でデータ件数が異なる．優先順位付けに関係なく，データ件数が大きくなると，最適な幅が大きくなる．データ件数が大きいほど，$score_1$ による高速化が大きくなる．

次に，Database 1、Database 2、および Database 3 の実験結果を比較することで，特徴データの次元数による影響を確認している．これらのデータベースは、異なる特徴次元数で同数のデータを持つ．データの特徴次元数が減少するにつれて，第 1 段階検索の候補数が減少し，最適なスケッチの幅が増加し，全探索に対する $score_1$ による高速化の効果が大きくなる．少ない次元数に含まれる距離情報は小さく，スケッチで簡単に解を得られるため、この効果は自然と考えることができる．

表 6.7　各画像データベースのスケッチの最適ビット幅

データベース	次元数	データ数	ハミング距離	$score_\infty$	$score_1$
Database 1	64	All	19	18	18
Database 2	36	All	19	20	19
Database 3	16	All	24	27	24
Database 4	64	1/4	17	16	16
Database 5	64	1/16	15	14	14

表 6.8　$score_1$ の列挙を用いた検索の高速化率

データベース			full search	提案手法（$score_1$）			高速化率
No.	次元数	データ数	(ms)	w	K(%)	(ms)	($\frac{\text{full}}{score_1}$)
1	64	All	551	18	1.51	4.97	111
2	36	All	331	19	0.653	1.31	253
3	16	All	136	24	0.132	0.179	759
4	64	1/4	136	16	2.37	2.02	67.4
5	64	1/16	34.1	14	3.00	0.615	55.4

表 6.9　実験結果（Database 1：データ数 = 6.9 $\times 10^6$, 特徴次元数 = 64）

優先順位付け	width	16	17	18	19	20	21
	$K(\%)$	6.82	6.32	5.60	5.25	4.89	5.01
	sketches	3676	6586	11271	20356	36181	71199
	empty	49	287	1189	4374	12608	35817
	elements	130	69	38	23	14	10
ハミング距離	time(ms)	18.8	18.1	17.0	**16.8**	17.2	19.7
	unsorted(ms)	62.0	58.5	53.3	50.9	49.2	52.5
	enum(ms)	0.697	0.695	0.718	0.858	1.08	1.55
	speedup	3.39	3.33	3.23	3.14	2.99	2.80
	$K(\%)$	4.59	4.32	3.77	3.70	3.32	3.02
	sketches	2407	4400	7383	13993	23944	40742
	empty	29	174	730	2899	8126	19943
	elements	133	71	39	23	14	10
$score_\infty$	time(ms)	12.7	11.9	**11.0**	11.7	11.6	12.3
	unsorted(ms)	38.3	36.5	32.5	32.5	30.2	28.4
	enum(ms)	0.470	0.494	0.498	0.612	0.763	0.978
	speedup	3.09	3.16	3.04	2.87	2.72	2.43
	$K(\%)$	2.03	1.75	1.51	1.37	1.22	1.33
	sketches	862	1377	2205	3630	5919	9640
	empty	4	26	119	454	1353	3453
	elements	163	89	50	30	18	13
$score_1$	time(ms)	5.74	5.22	**4.97**	5.15	5.56	6.50
	unsorted(ms)	18.5	16.4	14.3	13.8	13.3	13.5
	enum(ms)	0.309	0.360	0.472	0.699	1.11	1.82
	speedup	3.34	3.29	3.07	2.95	2.75	2.50

表 6.10　実験結果（Database 2：データ数 = 6.9 $\times 10^6$, 特徴次元数 = 36）

優先順位付け	width	16	17	18	19	20	21
	$K(\%)$	3.85	3.22	2.86	2.57	2.36	2.26
	sketches	1624	2472	4067	6583	10967	19047
	empty	122	370	1004	2431	5393	11575
	elements	177	106	64	43	29	21
ハミング距離	time(ms)	5.78	5.06	4.72	**4.43**	4.44	4.62
	unsorted(ms)	23.3	19.9	18.0	16.5	15.5	15.3
	enum(ms)	0.194	0.176	0.180	0.191	0.226	0.283
	speedup	4.13	4.05	3.92	3.85	3.64	3.47
	$K(\%)$	2.03	1.75	1.60	1.40	1.29	1.17
	sketches	725	1121	1859	2826	4618	7360
	empty	39	128	365	860	1946	3963
	elements	204	122	74	49	33	24
$score_\infty$	time(ms)	3.12	2.86	2.72	2.51	**2.50**	2.54
	unsorted(ms)	11.77	10.48	9.62	8.62	8.18	7.59
	enum(ms)	0.106	0.0980	0.103	0.106	0.124	0.146
	speedup	3.86	3.76	3.64	3.55	3.38	3.11
	$K(\%)$	0.974	0.839	0.737	0.653	0.586	0.518
	sketches	252	370	557	813	1227	1790
	empty	5	19	56	142	330	667
	elements	272	165	101	67	45	32
$score_1$	time(ms)	1.59	1.44	1.32	**1.31**	1.36	1.41
	unsorted(ms)	5.90	5.26	4.75	4.38	4.13	3.88
	enum(ms)	0.0758	0.0868	0.108	0.144	0.206	0.300
	speedup	3.86	3.82	3.83	3.64	3.39	3.22

表 6.11　実験結果（Database 3：データ数 = 6.9 $\times 10^6$, 特徴次元数 = 16）

優先順位付け	width	20	22	24	26	27	28
	$K(\%)$	0.834	0.669	0.538	0.505	0.497	0.511
	sketches	1190	2526	5154	13851	23512	41555
	empty	743	1909	4380	12746	22191	39940
	elements	129	75	48	32	26	22
ハミング距離	time(ms)	0.696	0.637	**0.606**	0.729	0.836	1.04
	unsorted(ms)	2.61	2.23	1.92	1.98	2.09	2.34
	enum(ms)	0.0496	0.0523	0.0678	0.122	0.176	0.275
	speedup	3.95	3.72	3.44	3.06	2.89	2.69
	$K(\%)$	0.301	0.239	0.191	0.158	0.139	0.130
	sketches	172	326	590	1190	1657	2421
	empty	59	164	380	921	1362	2092
	elements	184	102	63	40	33	27
$score_\infty$	time(ms)	0.269	0.237	0.215	0.205	**0.204**	0.209
	unsorted(ms)	1.02	0.845	0.718	0.635	0.586	0.573
	enum(ms)	0.0190	0.0191	0.0212	0.0276	0.0323	0.0404
	speedup	3.99	3.80	3.60	3.42	3.23	3.16
	$K(\%)$	0.229	0.172	0.132	0.107	0.0990	0.0901
	sketches	90	144	226	400	550	704
	empty	18	47	105	244	370	507
	elements	219	122	75	47	38	32
$score_1$	time(ms)	0.216	0.196	**0.179**	0.206	0.231	0.265
	unsorted(ms)	0.821	0.670	0.563	0.525	0.533	0.541
	enum(ms)	0.0234	0.0300	0.0429	0.0739	0.103	0.136
	speedup	4.14	3.85	3.83	3.40	3.36	3.14

表 6.12　実験結果（Database 4：データ数 = 6.9 $\times 10^6/4$, 特徴次元数 = 64）

優先順位付け	width	15	16	17	18	19	20
	$K(\%)$	8.95	8.21	6.92	6.86	6.97	7.23
	sketches	2565	4564	7356	14396	28223	56808
	empty	27	175	806	3313	10914	31267
	elements	243	129	73	43	28	20
ハミング距離	time(ms)	6.53	6.22	**5.65**	6.18	6.99	8.41
	unsorted(ms)	22.48	20.90	17.93	18.39	19.41	21.31
	enum(ms)	0.254	0.271	0.280	0.407	0.620	0.966
	speedup	3.54	3.47	3.29	3.12	2.95	2.73
	$K(\%)$	5.81	5.25	4.58	4.22	4.00	3.86
	sketches	1634	2882	4797	8587	15581	28629
	empty	16	109	517	1906	5882	15333
	elements	248	131	74	44	28	20
$score_\infty$	time(ms)	4.26	**4.06**	3.77	3.95	4.35	4.88
	unsorted(ms)	13.84	12.72	11.33	10.81	10.63	10.84
	enum(ms)	0.166	0.176	0.206	0.263	0.381	0.555
	speedup	3.34	3.23	3.12	2.86	2.58	2.38
	$K(\%)$	2.83	2.37	2.07	1.83	1.61	1.37
	sketches	679	1074	1729	2847	4556	6839
	empty	3	22	116	437	1278	2882
	elements	289	156	89	53	34	24
$score_1$	time(ms)	2.26	**2.02**	2.04	2.24	2.58	3.01
	unsorted(ms)	7.17	6.34	5.81	5.41	5.41	5.40
	enum(ms)	0.158	0.208	0.297	0.476	0.777	1.195
	speedup	3.33	3.39	3.16	2.80	2.56	2.31

表 6.13　実験結果（Database 5：データ数 = 6.9 $\times 10^6/16$, 特徴次元数 = 64）

優先順位付け	width	13	14	15	16	17	18
	$K(\%)$	10.50	8.83	8.37	7.80	7.58	7.84
	sketches	774	1274	2383	4335	8195	16703
	empty	1	18	142	158	2428	7819
	elements	937	485	258	146	91	61
ハミング距離	time(ms)	1.92	1.69	**1.67**	1.69	1.84	2.17
	unsorted(ms)	5.48	4.70	4.56	4.37	4.41	4.87
	enum(ms)	0.0739	0.0721	0.0888	0.118	0.176	0.298
	speedup	2.93	2.87	2.82	2.71	2.55	2.44
	$K(\%)$	6.84	6.35	5.93	5.50	4.80	4.67
	sketches	496	901	1646	2974	5040	9434
	empty	1	12	92	432	1451	4293
	elements	954	493	263	149	92	63
$score_\infty$	time(ms)	1.26	**1.21**	1.21	1.24	1.22	1.42
	unsorted(ms)	3.50	3.29	3.14	3.02	2.77	2.86
	enum(ms)	0.0515	0.0551	0.0658	0.0880	0.122	0.191
	speedup	2.86	2.81	2.68	2.54	2.41	2.18
	$K(\%)$	3.66	3.00	2.67	2.39	2.14	1.85
	sketches	239	372	627	1067	1792	2844
	empty	0	2	21	108	392	1045
	elements	1057	560	304	172	106	71
$score_1$	time(ms)	0.728	**0.615**	0.656	0.696	0.821	1.01
	unsorted(ms)	1.97	1.66	1.56	1.54	1.56	1.62
	enum(ms)	0.0513	0.0653	0.100	0.170	0.273	0.451
	speedup	2.83	2.91	2.62	2.60	2.36	2.09

第 7 章

結論と今後の課題

スケッチを 32 ビットからより狭い 16 ビットにし，効率的な第 1 段階検索とバケット法によるデータ管理によって約 10 倍の高速化を実現した．今回，16 ビットスケッチに対してバケット法を用いるだけでなく，質問のスケッチに近い順にスケッチを列挙することによる第 1 段階検索の高速化を行った．

これにより，最速の検索方法が考案された．この論文で提案された $score_1$ 順の列挙コストは十分に小さいが，ハミング距離または $score_\infty$ 順の列挙コストほど無視できない．したがって，状況によっては，$score_1$ による検索速度が $score_\infty$ による検索速度よりも遅くなる場合がある．たとえば，27 ビット以上のスケッチの場合，$score_1$ による検索時間は $score_\infty$ による検索時間より長くなる．$score_1$ の列挙アルゴリズムには $O(\log n)$ の遅延があり，$score_\infty$ の列挙アルゴリズムは $O(1)$ で極めて小さな定数遅延であるためである．$score_1$ のより効率的な列挙アルゴリズムを見つけることが重要な課題である．

距離下限値に基づくスコア $score_1$ と $score_\infty$ の課題は，ハミング距離よりも正確な優先順位付けになる理由を調査することである．また，$score_2$ は，文献 [7] で $score_1$ とほぼ同精度であることが報告されているので，本論文では用いなかったが，たとえば，$score_{1.5}$ などとともに検討する必要があるかもしれない．

幅の広いスケッチは，大規模なデータベースまたは低次元のデータベースに適していることを示した．たとえば，特徴次元数 64 の約 690 万件の画像データベース（Database 1）の場合，18 ビットまたは 19 ビットのスケッチを使用した検索が最も高速である．次元数 16 の Database 3 の場合，24 ビットまたは 27 ビットのスケッチが最適である．したがって，より広いスケッチが適している可能性がある．ただし，非常に幅の広いスケッチを使用すると，空のバケットが非常に多くなるため，バケット方式を効率的に使用できない．したがって，より広いスケッチのデータ管理を検討する必要がある．最初に狭い 16 ビットスケッチを使用する方法に焦点を当てた理由の 1 つは，スケッチの総数がデータベースサイズに比べて少なく，すべてのスケッチが検索されたとしても，第 1 段階検索が小さい一定のコストで済む．しかしながら，優先順序付けを用いたスケッチの効率的な列挙を使用すると，約 690 万件のデータベースに対して 24 ビットおよび 28 ビットのスケッチを使用して高い性能を得ることができる．Database 3 を使用した実験結果によると，28 ビットスケッチを使用した場合，空のバケットが増加してもソートによる高速化効果は維持される．これは，検索時にアクセスされた空でないバケットの平均要素数が比較的多いという事実によるものと思われる．この現象は，高性能のスケッチを低次元データの LSH として作成できるためだと考えられる．したがって，高次元データに対してより高い LSH

性能を持つスケッチを作成する方法について検討することは，今後の重要な作業である．焼きなまし法の一種である AIR を使用すると，QBP よりも衝突確率が小さいスケッチのピボットセットを取得できるが，検索の精度は向上しない [11]．スケッチの最適化をさらに調査する必要がある．

参考文献

[1] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proc. VLBD'97*, pp. 426–435, 1997.

[2] W. Dong, M. Charikar, and K. Li. Asymmetric distance estimation with sketches for similarity search in high-dimensional spaces. In *Proc. ACM SIGIR'08*, pp. 123–130, 2008.

[3] C. Faloutsos and K.I. Lin. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proc. 24th ACM SIGMOD'95*, pp. 163–174, 1995.

[4] A. Guttman. R-trees: A dynamic index structure for spatial searching. In Beatrice Yormark, editor, *Proc. SIGMOD'84*, pp. 47–57, 1984.

[5] 樋口直哉. 焼きなまし法を用いた縮小構造 sketch の最適化と多次元データ近似検索の高性能化に関する研究. 九州工業大学大学院情報工学府修士論文, 2016.

[6] N. Higuchi, Y. Imamura, T. Kuboyama, K. Hirata, and T. Shinohara. Nearest neighbor search using sketches as quantized images of dimension reduction. In

Proc. ICPRAM'18, pp. 356–363, 2018.

[7] N. Higuchi, Y. Imamura, T. Kuboyama, K. Hirata, and T. Shinohara. Fast filtering for nearest neighbor search by sketch enumeration without using matching. In *Proc. AI'19*, pp. 240–252, 2019.

[8] N. Higuchi, Y. Imamura, T. Kuboyama, K. Hirata, and T. Shinohara. Fast nearest neighbor search with narrow 16-bit sketch. In *Proc. ICPRAM'19*, pp. 540–547, 2019.

[9] N. Higuchi, Y. Imamura, T. Kuboyama, K. Hirata, and T. Shinohara. Annealing by increasing resampling. In *In Selected papers from ICPRAM'19, LNCS, Springer (to appear).*

[10] 樋口直哉, 今村安伸, 久保山哲二, 平田耕一, 篠原武. 狭い 16 ビットのスケッチを用いた高速最近傍検索. 情報処理学会論文誌「数理モデル化と応用」（採録決定）.

[11] Y. Imamura, N. Higuchi, T. Kuboyama, K. Hirata, and T. Shinohara. Pivot selection for dimension reduction using annealing by increasing resampling. In *Proc. LWDA'17*, pp. 15–24, 2017.

[12] Y. Imamura, N. Higuchi, T. Shinohara, T. Kuboyama, and K. Hirata. Annealing by increasing resampling in the unified view of simulated annealing. In *Proc. ICPRAM'19*, pp. 173–180, 2019.

[13] 岩崎瑶平. 座標分割法を用いた局所性鋭敏ハッシング (LSH) による近似検索の高性能化に関する研究. 九州工業大学大学院情報工学府修士論文, 2010.

[14] 岩崎瑶平, 大野真吾, 田島圭, 篠原武. 座標分割法を用いた Locality Sensitive Hashing による近似検索の高性能化に関する研究. 人工知能学会人工知能基本問題研究会 (第 81 回) 資料, pp. 69–74, 2011.

[15] 児玉晋, 篠原武. Hilbert R-tree を用いた一括検索処理の高速化のためのノード訪問順に関する研究. 人工知能学会人工知能基本問題研究会 (第 93 回) 資料, pp. 37–43, 2014.

[16] V. Mic, D. Novak, L. Vadicamo, and P. Zezula. Selecting sketches for similarity search. In *Proc. ADBIS'18*, pp. 127–141, 2018.

[17] V. Mic, D. Novak, and P. Zezula. Improving sketches for similarity search. In *Proc. MEMICS'15*, pp. 45–57, 2015.

[18] V. Mic, D. Novak, and P. Zezula. Designing sketches for similarity filtering. In *ICDMW'16*, pp. 655–662, 2016.

[19] V. Mic, D. Novak, and P. Zezula. Speeding up similarity search by sketches. In *Proc. SISAP'16*, pp. 250–258, 2016.

[20] V. Mic, D. Novak, and P. Zezula. Sketches with unbalanced bits for similarity search. In *SISAP'17*, pp. 53–63, 2017.

[21] V. Mic, D. Novak, and P. Zezula. Binary sketches for secondary filtering. *ACM Trans. Inf. Syst.*, Vol. 37, No. 1, pp. 1:1–1:28, 2018.

[22] V. Mic, D. Novak, and P. Zezula. Modifying hamming spaces for efficient search. In *ICDMW'18*, pp. 945–953, 2018.

[23] A. Müller and T. Shinohara. Efficient similarity search by reducing i/o with compressed sketches. In *Proc. SISAP'09*, pp. 30–38, 2009.

[24] G. Navarro. Searching in metric spaces by spatial approximation. *The VLDB Journal*, Vol. 11, No. 1, pp. 28–46, 2002.

[25] T. Shinohara and H. Ishizaka. On dimension reduction mappings for approximate retrieval of multi-dimensional data. In *Progress of Discovery Science, LNCS 2281, Springer*, pp. 89–94, 2002.

[26] Z. Wang, W. Dong, W. Josephson, M. Charikar Q. Lv, and K. Li. Sizing sketches: A rank-based analysis for similarity search. In *Proc. ACM SIGMETRICS'07*, pp. 157–168, 2007.

[27] P.N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proc. SODA'93*, pp. 311–321, 1993.

[28] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity search: The metric space approach.* Advances in Database Systems. Springer, 2006.